University of Edinburgh
Division of Informatics

Colour Management and Colour Correction in CinePaint

4th Year Project Report
Computer Science

Stefan Klein

May 26, 2004

Abstract: Before the advent of affordable desktop computers, colour reproduction used to be the task of trained specialists. As it moved to the desktop and into the reach of everyone, systems failed to provide the means to handle it well. Users are still surprised when they find that colours do not appear the same on all screens or printers. To provide means of reliable colour reproduction is the goal of colour management.
Colour correction or colour grading refers to the manipulation of colours. In film, grading gives a film a consistent look. It can set an atmosphere for a scene, for example by bathing it in warm evening light. It can also be a means of artistic expression, such as symbolising the loss of joy of a film character through the use of bleached colours.

This project is set within the open source project CinePaint, an image retouching application that has been used for films such as Harry Potter and The Last Samurai. The goal is to equip CinePaint with a display colour management system. This will make CinePaint the second open source application to implement such a system.
The goal is further to provide tools for colour correction, namely a colour corrector based on colour wheels and look profiles. Whilst colour wheels are a widely used concept, look profiles are only starting to be explored by developers of professional applications.

Acknowledgments

The author would like to thank the following:

Robin Rowe at CinePaint for inviting me to work on the CinePaint project, for invaluable advice, in particular during the early stages of the project, and for many interesting discussions.

Julian Bradfield for making this project possible in the first place by supervising me.

Kai-Uwe Behrmann at CinePaint for useful feedback and cooperation.

John Matthews at the BBC for taking the time to show me his work in spite of a very busy schedule.

Contents

1 Introduction....7
1.2 Motivation....7
1.3 CinePaint....10
1.4 Goals....10
1.5 Results....11
1.6 Remaining Chapters....11
2 Background....13
2.1 Describing colours....13
2.2 Matching - mixing colours from colours....14
2.3 Physical limits - problems with matching....15
2.4 Colour Management....16
2.5 Visual Colour Models and colour wheels....17
2.6 Colour Correction....18
3 Design....21
3.1 Fundamental design decisions....21
3.1.1 A format for colour management conversions: ICC profiles....22
3.1.2 A colour management module: littleCMS....24
3.1.3 Handling many profiles: Storage and Caching....25
3.1.4 A format for look profiles: ICC abstract profiles....25
3.2 Aside: familiarisation with the code base - binary fixed point....26
3.3 CinePaint internals....27
4 Implementation I: Colour Management....29
4.1 Challenges....29
4.2 The User's View....30
4.2.1 Configuration....30
4.2.2 Use....32
4.3 Behind the scenes....32
4.3.1 Efficiency I - The Profile cache....32
4.3.2 Efficiency II - Display buffer....33
4.3.3 Fault Tolerance....34
5 Implementation II: Colour Correction....35
5.1 Challenges....36
5.2 The user's view....36
5.3 Behind the Scenes....39
5.3.1 Drawing the colour wheel....39
5.3.2 Colour correction....40
5.3.3 Look Profiles....41
6 Evaluation....43
6.1 Colour Management....44
6.1.1 Interface Justification....44
6.1.2 Testing the Implementation....44
6.2 Colour Correction....45

6.2.1 Interface Justification.....45
6.2.2 Testing the Implementation.....45

7 Conclusion.....49
7.1 Achievements.....49
7.2 Scope for future work.....50

References

Appendix

INTRODUCTION, n. A social ceremony invented by the devil for the gratification of his servants and the plaguing of his enemies

Ambroce Bierce, US Journalist

1 Introduction

The goal of this project is to provide means of reliable colour reproduction and colour manipulation for the open source image retouching program CinePaint.

1.1 Motivation

Colours are an important means of communication and expression. Colours convey structure in diagrams, they create emotions and atmosphere. The French artist Yves Klein (1928-1962) developed a major part of his works on one single blue that he created specifically and which he patented as International Klein Blue (IKB). He claimed it to be the physical representation of cosmic energy that, otherwise invisible, floats in the air freely. People have favourite colours, companies have corporate colours. Colour photography, the first colour television, and more recently the breakthrough of colour in newspaper caused excitement and were usually seen as a

big improvement over their black and white predecessors. Colour plays an important part in the world, above all in art and design. Colours are chosen specifically - a costumer wants an arm chair in azure blue, not in royal blue or in sky blue. The difference between a dull photograph and one with vivid colour that brings back the memories of when it was taken may be subtle, but it makes all the difference. In short, colour matters.

Colour Management

Colour reproduction in computer graphics, however, has for a long time been of very low quality. Before the advent of affordable desktop computers, colour reproduction used to be the task of trained specialist using expensive equipment. But then it moved into the reach of people who had no background in colour theory and who were given the impression that they had the freedom to use any colour they wanted and that the print would be an exact match of the image on the screen. Many developers probably believed it themselves.

However, it has long been known that this must be an illusion. Colorimetry, the physics of colours, made great advances over the last century. The response of the human visual system to colour was quantified through the colour matching experiments. Sophisticated colour models were developed by the CIE, the Commision International de L'Eclairage (International Commission on Illumination, www.cie.co.at/cie), an international body devoted to cooperation and exchange in the science and art of lighting.

Here is the bad news that comes as a surprise to many: Your monitor cannot display all the colours you can see. The set of colours a monitor can reproduce differs from model to model, and the same colours might appear differently on a different monitor. The same colour is most likely to appear differently on a printer.
This arises from the difficulty to communicate colours clearly. Different devices such as printers or monitors use different ways of reproducing colours. It is as if they were speaking different languages. Still they all pretend to understand each other. The result are many misunderstandings (a more factual explanation can be found in chapter 2).

Eventually, however, software developers realised that they had to provide solutions to the problem in the form of colour management systems (CMS). *Apple* was first to release its *ColorSync* (www.apple.com/macosx/features/colorsync/) colour management system in 1991 as part of its operating system *System 7*. In 1993, the *International Colour Consortium* (ICC, www.color.org), founded by several computer manufacturers (Adobe, Agfa-Gevaert, Apple, Kodak, Microsoft, Silicon Graphics, Sun Microsystems, Taligent) and FOGRA (the German graphics arts research institute), took up work with the goal to create and promote the standardisation of open colour management systems. *Microsoft* introduced a first version of its colour management system with the release of *Windows95* (*ICM*, http://www.microsoft.com/whdc/device/display/color/icmwp.mspx). And finally, in 1998 *Adobe Photoshop 5* was the first graphics package to provide full colour management functionality. The users, however, had not gone through the same process of realisation as the developers. The release of *Photoshop 5* caught them unprepared and caused a lot of confusion, often resulting in users turning colour management off. Today, however, at least in the professional world, colour management has finally become the serious issue it should have been from the outset.

In the domain of open source software colour management is still underdeveloped. Linux does not provide a CMS on operating system level. However, there is a very mature and reliable library for colour conversions (LittleCMS, www.littlecms.com). *Scribus*, a desktop publishing program (www.scribus.net) is at the time of this writing the only open source application to fully implement colour management.

Users want predictable and consistent colours. They want the colours they see on the screen to

match the colours on any screen or the colours they finally print. Modern graphics applications are based on the principle that they provide the user with a preview of the final result. This should include colours.
Without reliable colour preview on monitors, proof prints are an absolute necessity and cause extra costs. The more reliable colour preview on monitors is, the less proof prints will be needed.

There is a need for more applications to adopt colour management, especially in the open source sector, both to improve quality and reliability and to further the understanding of colour handling amongst users and developers of such applications

Colour Correction

Ideally, reliable colour reproduction would go without mentioning for any professionally used graphics application. Their real focus lies on providing means of manipulating images. One aspect of this is manipulating their colours.

In film, the work that is centred around colour is carried out by a colourist and called colour correction or colour grading. His task is two-fold: One is to make the film look consistent. The shooting of a scene may take place under changing light conditions. For example, it may be partly shot outside and partly on set. If in the story the scene is meant to take place in a single location, the colourist has to even out differences in lighting, otherwise the editing will appear uneven.
The other task is more creative. It is to give the film a look and to create colour effects: A blueish futuristic look of *Minority Report* with *Tom Cruise.* Epic scenes that look like great wall paintings like in *The Lord of the Rings.* The impression of an old technicolor film**.** Faded colours, very strong colours or overexposed colours to express the particular state of mind of a character. The examples are endless.

It is often not easy to spot the particular problems or needs for correction of a given image. The difference between a dull photo and a vivid one that brings back all the memories of where it was taken is often subtle. The human visual system (HVS) is very good at evening out errors in images. A white shirt in an image may be slightly pink due to a wrong camera setting, still at first sight, the brain will suggest otherwise, because it *knows* that the shirt is white. This phenomena is called colour constancy, the ability of the HVS to make colours appear the way they should. This often makes it very difficult to judge an image. Take a look at the tow images in figure 1 of the appendix (and only this figure). They look presentable. Then take a look at figure 2 on the following page. You will suddenly notice that the previous images were not very good at all, that they had fairly strong yellow and red casts on them. Still, without experience, it takes most people the comparison between the two pictures to notice it clearly.

Colours are not easy to get firm control over. Changing colours is much harder than changing the formatting in a word processor. Sometimes a little tweaking makes a big difference, sometimes a huge shift does not get you anywhere. This stresses the importance of appropriate tools. They need to afford intuitive manipulations. They need to contain in their design some measure of what can be done and what might have to be done to correct a given image.
Suitable tools provide handles onto colour in abstract ways. Artists do usually not think in terms of the red, green and blue that a monitor works with. They work with abstractions such as hue and saturation and they need ways of inspecting and manipulating them effectively and reliably.

Common classes of tools can be found in many professional software packages such as *Adobe Photoshop, Discreet Combustion, Apple Final Cut Pro or Apple Shake*. Two of the most popular ones are colour curves that describe colour changes by visually manipulating a mathematical curve relating input colour to output colours and colour wheels that allow to change the colour balance of an image shifting a handle over a colour wheel. This project will

focus on colour wheels.

With an increasing number of films being edited digitally, the possibilities of artistic expression are increasing, too. An increasingly used concept is *look management* which allows storing colour corrections in data structures called *look profiles* which can then be used to preview or to apply the same corrections on other images. This allows the colourist to create a collection of looks to experiment with. Studios currently use their own tools to achieve this. There are no publicly available implementations, yet, as far as I am aware. Kodak was the first to present a Look Management System at the American National Association of Broadcaster's (NAB) Spring Convention in April this year. It is, however, focused on simulating the look of their different film materials rather than on artistic looks.

Advances in colour correction tools are unfortunately often made in the proprietary contexts of editing studios and software companies and the techniques that make one implementation of a correction tool superior stay secret. CinePaint is the first package that is open and accessible while aiming at achieving the same high-quality standard in image editing as professional commercial packages. It makes them available to a wide audience offering new ways of artistic expression. If it succeeds in its goal to provide high quality implementations of standard image manipulation techniques, it will also be an interesting source for study. A graphics package is a complex application and much can be learned from it. At the moment that information is difficult to access.

1.2 CinePaint

CinePaint is used by a range of studios and played a part in the post production of films such as *Harry Potter*, *The Last Samurai*, *Stuart Little II* and *2 Fast, 2 Furious*.
It is a painting and image retouching program which started in 1998 as a branch of the GIMP package which was known as filmgimp. It was, however, never built into a GIMP release and almost forgotten about. In July 2002 Robin Rowe wrote an article for the Linux Journal (linuxjournal.com) about its use at the Rhythm&Hue film studios. Subsequently, Rowe decided to revive the project as a sourceforge project and became its project leader. Filmgimp became CinePaint.
CinePaint focuses on professional features such as high colour depth or high dynamic range and on film-specific features such as a flipbook that allows to easily load, store and move through sequences of images.

1.3 Goals

The aim of this project is to equip CinePaint with full display colour management on the basis of the ICC standard. The purpose of display colour management is to maintain colour appearance the same between different display devices, such as a CRT and a TFT. This project is not concerned with colour management in printing, but the results will be readily extensible. The reason printing was disregarded is that it is probably almost harder to set up a good demonstration than to do the actual implementation.
Within the display CMS it will be possible to specify image and monitor characteristics in the form of ICC profiles (which will be explained in 3.1.1) and to correctly reproduce colours on any display. It will be possible to convert between colour representations. Considering that the conversions needed to correct the colours will cause an extra time strain on the work of the rendering engine, special attention will be paid to efficiency.

The aim is further to provide a new colour correction tool to CinePaint. CinePaint already has a

curves tool which was inherited from the GIMP, but does not have a colour corrector based on colour wheels. The goal is to provide such a colour corrector. This involves designing an accessible interface and working out an underlying algorithm. The corrector will be designed with a future integration with the curves tool in mind. Integrating the two will equip CinePaint with a single powerful colour correction tool.

Finally the colour correction tool will be complemented by look profiles. They will provide the possibility to store manipulations in a separate file called a look profile and allow previewing images through the profile without affecting the image data. At the same time, they will allow committing the manipulations to the image data, meaning permanently changing the image data in accordance with the profile. They should be seen as the first step for CinePaint towards providing its own full Look Management.

1.4 Results

The system specified above has been fully implemented.

The display colour management was implemented particularly efficiently using a profile cache and will form a robust foundation for further developments in the area such as printer colour management.

The exact requirements for the colour wheels were difficult to establish which led to the development of a range of smaller variations and user options room for interesting comparisons and fine tuning.

Look profiles were implemented as anticipated.

1.5 Remaining Chapters

The remainder of this report is divided into chapters as follows:

Chapter 2 gives the necessary background on colour theory and colour models and explains why colour reproduction cannot be reliable without a colour management system. It concludes with an overview of colour correction.

Chapter 3 describes the formats and libraries that were used in the design of this project and explains their choice. It also gives some detail on the existing codebase of CinePaint.

Chapter 4 and 5 explain the work done in this project in detail. Chapter 4 focuses on Colour Management whilst chapter 5 covers the colour wheels colour corrector and look profiles.

Chapter 6 outlines the steps that were taken to ensure the usefulness and correctness of the implementation of this project

Chapter 7 finally summarises the achievements and sets the scope for possible future work.

The *Appendix* contains a few colour illustrations.

One's mind has a way of making itself up in the background, and it suddenly becomes clear what one means to do.

A.C. Benson

2 Background

This chapter gives the necessary background on colour theory and colour models and explains why colour reproduction cannot be reliable without a colour management system. It concludes with an overview of colour correction.

2.1 Describing colours

In order to be able to store information about colours and to reproduce colours from this information, reliable ways of describing them are necessary.

colour charts

The earliest approach to classifying colours was to use nature as a reference by naming colours after organic compounds or plants that exhibited them most distinctively. This is still reflected in common colour names such as orange, rose, violet or azure. The foremost colour researches of these days were natural scientists who developed colour atlases containing reference patches

tagged with a colour name and a list of flowers or insects that exhibit it. Today, the same principle is still widely used in printing and design where colours are often specified as references to standardised and often patented colour charts. The pantone system is a common example.

colour attributes

Artists searched for more generic ways of describing colour in their quest to gain control over them and to use them for their purposes. One such way of describing the visual sensation of a colour is by the three attributes hue, saturation and lightness. Hue is the most familiar attribute. It refers to the basic quality of a colour. It is what leads us to label a colour with a common name such as orange, green or blue. Saturation is a measure of a colours purity or intensity. Saturated colours such as royal blue appear intenser, whilst unsaturated colours such as sky blue appear paler. Finally, lightness expresses how light or dark a colour appears.

Both colourcharts and descriptions in terms of hue, saturation and lightness are subjective ways of specifying colours based on visual impressions. They are appropriate for art and design, in a technical context, however, more precise specifications are needed.

Such specifications started to develop with the beginning of the scientific study of light and the the human visual system (HVS) by Newton in 1704 which eventually spawned a branch of physics known as colorimetry.

physical specifications

Physicists discovered that light fundamentally is electromagnetic waves with wavelengths ranging from 400nm to 700nm, that most light contains a wide spectrum of different wavelengths and that it is the wavelengths that determine the colour impression. They realised that there were physical correspondences to hue, saturation and lightness. The hue is the dominant wavelength in a colour's spectrum. Saturation corresponds to what is called the excitation purity of a spectrum measuring how little or how many different wavelength it contains. Lightness arises from the luminance of a light source. Luminance is the light energy a source emits weighted by a spectral weighting curve. The weighting curve reflects the fact that the HVS perceives some wavelength as brighter than others. A blue light will appear darker than a yellow light, even though the light energy emitted may be the same.
Luminance and lightness are easily confused. They are, however, not the same. Luminance refers to the physical brightness of a light source, lightness to how this brightness is perceived by the HVS. This perception is non-linear: A light source with only 18% luminance of a reference light source, will appear to the HVS as half as bright.

Since dominant wavelength, excitation purity and luminance are physical quantities, they allow a precise specification of a colour. Unfortunately this is not practical, since these quantities are hard to measure.

2.2 Matching - mixing colours from colours

a simpler colour specification

Artists, however, have long known that you can create new colours by mixing existing ones. Biology discovered that the human retina contains three different kinds of colour sensors, the so-called cones, that exhibit sensitivity to different wavelength with peak sensitivity at approximately red, green and blue respectively (the so-called tristimulus theory). From these three sensor responses the brain is able to create all possible colour sensations.

This suggests an attractive way of specifying colours: as a weighted mixture of base colours, as a colour recipe if you like (take two measures of red, three of...). The base colours are referred to as primaries and the most commonly used primaries are red, green and blue due to how the HVS is built. The process of defining a colour by a weighted sum of three primaries is called colour-matching and the science around it is called trichromatic (meaning 'three-coloured')

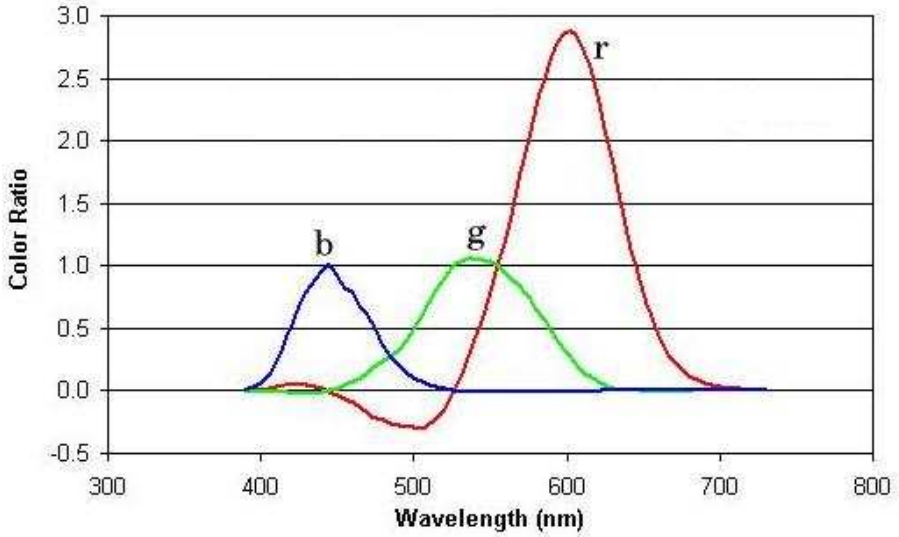


Figure 2.1 The rgb colour-matching curves

colorimetry. All modern display devices such as TVs or monitors are based on this principle. They mix colours from varying proportions of red, green and blue.

matching experiments

Experiments were conducted to further quantify the responses of the three cones. In these experiments, a light source emits light of a single wavelength producing a coloured reference patch on a wall. Next to it, three sources of red, green and blue light overlap to form a second patch. The subject of the experiment is asked to adjust the intensities of the three overlapping sources such that the two colour patches look the same. This is repeated setting the reference patch one by one to all wavelengths of the visible spectrum. Repeating the experiment with many subjects lead to the derivation of what is known as the rgb colour-matching curves (Figure 2.1). The curves express how much red, green and blue have to be used to mix a given colour.

The experiments discovered, however, that certain colours could not be matched at all. For certain colours of the reference patch it was impossible to find settings of the three mixing light sources such that the mixed patch would be identical. The only way of achieving identity was to modify the reference patch itself, to add a little red to it. Mathematically this corresponds to adding negative red to the mixed patch which is reflected in the curves.

This discovery poses a serious problem for computer graphics: Colours on a monitor are generated by mixing red, green and blue. But we just saw that it is impossible to mix all colours the HVS is capable of perceiving! Even worse, different sets of primaries, of base light sources, will generate different sets of colours when combined. The set of all colours that can be mixed from a given set of primaries is called their colour space or colour gamut.

2.3 Physical limits - problems with matching

This is where the problem with colour reproduction in computer graphics starts: Usually colours are stored in files as proportions of red, green and blue light which are applied to the three light sources of a monitor to reproduce a colour according to the stored specification. However, there is no guarantee that the red, green and blue on one monitor is the same as on another. Admittedly, the differences between different CRT monitors are subtle, but noticeable.

The differences become more obvious, if you compare CRT monitors to TFT displays.

You may compare it to cooking. If you follow the exact same recipe (i.e. if use the exact same colour specification), but in one case you use fresh vegetables, in the other tinned vegetables (i.e. if the primaries change), the outcome will be different. Colours will not necessarily appear the same on each monitor if they are simply specified as proportions of its red, green and blue light sources.

When you go to a department store, compare the TVs set up on its shelves. They all show the same program, they all receive the exact same input signal, but you will easily observe that the colours can appear quite differently from set to set.

This states the first problem: *Storing colours as red, green and blue values is not an accurate colour description. The same colour can appear different on different monitors.*

The fact that no set of three primaries can represent all colours the HVS is capable of perceiving causes the second problem: *If a colour can be reproduced on one monitor, it does not mean it can be represented on any other monitor.*

This is a very common problem in printing. Printers use a colour model that is altogether different from monitors (a substractive instead of an additive model, I will not go into details, since they are not relevant here). The result is that many colours you can see on your screen, can never be printed the same way.

Finally, non-linearity of device input signals influences colour appearance. An example is a monitor's gamma: Applying red, green and blue input signals to a monitor usually assumes that the relation between input value and brightness was linear. If an input of 255 corresponds to maximum brightness, 127 should be 50% brightness, 64 should be 25%. It is not. The relation between a monitor's input signal and the generated brightness is a power function. The numerical value of the exponent of this power function is called the monitor's gamma. Gamma for a typical CRT is around 2.5, which gives approximately $intensity = input^{2.5}$ This means that an input of 127 (50%) actually only corresponds to a brightness of 17%!

Evening out all these problems is the task of colour management.

2.4 Colour Management

The first step in tackling this is finding an appropriate colour space. The RGB space is obviously inadequate, because colours specified in this space depend on the device they are reproduced on. Such gamuts, such colour spaces are called device-dependent. What is needed is a device-independent space. And it has to be a gamut that can represent all colours the HVS is capable of perceiving.

The colour-matching experiments showed that it is impossible to match any visible colour by a weighted sum of red, green and blue, because it would require negative light.

However, the fact that negative luminance cannot exist in reality, does not prevent assuming its existence, just as a fourth and fifth dimension can be assumed. The colour-matching functions could then be used to describe any colour, even if it cannot physically be generated on the three sources the colour matching functions are extracted from, because it would require negative luminance.

XYZ, a new colour model

It seemed awkward, though, to assume the existence of negative luminance. Besides, the problem that there is no clear definition of the primaries persists: The exact red, green and blue used to reproduce a colour depend on the monitor that is used. Therefore in 1931, the Commision International de L'Eclairage (International Commission on Illumination, CIE, www.cie.co.at/cie), an international body devoted to cooperation and exchange in the science and art of lighting, defined three standard primaries called X, Y and Z to replace red, green and blue in the matching process. They are positive over the whole visible spectrum and they can be used to describe all visible colours. Obviously, they are merely mathematical constructions and do not correspond to real light sources. One could not build a light that emits X, Y or Z light. Their definition is still useful, because any combination of real light sources can be translated into XYZ components. So they provide a way to identify a colour unmistakably and independently of any specific device or light sources.

rendering intents

However, to actually display an XYZ colour, it obviously has to be converted it into a mixture of any real R, G and B light sources. This may mean that the conversion is not possible, because the particular colour specified in XYZ cannot be matched by the red, green and blue of a given monitor. In this case, a way of approximating the colour is needed. This may mean clipping colours outside the gamut to the nearest colour inside. Or modifying the colour trying to maintain certain qualities like its saturation. The different approaches to approximation are known as *rendering intents* in the context of colour management.

conversion with profiles

To convert between RGB and XYZ, a conversion function is needed. This function depends on the primaries of the particular device that is used for colour reproduction. In colour management, such functions are usually stored in a file called a *colourmanagement profile.*

Commonly two profiles are needed in a complete colourmanagement conversion: assume we had an image with colours specified in RGB. With the image, we have got a profile describing how to convert the RGB specifications in the image into XYZ. This profile could, for example, be the profile of the scanner that the image was sampled on. We then have a second profile describing the RGB specifications of our monitor. The two profiles can be used to covert from image RGB to XYZ using the first profile and then from XYZ to monitor RGB using the second profile. The two sets of RGB values before and after the complete conversion are likely to be different. They are different, because they take into consideration the differences between the scanner and the monitor, but thanks to the conversion they correspond to the same XYZ colour specification, to the same physical colour.

This is the essence of colour management.

2.5 Visual Colour Models and colour wheels

In practice there is a whole wealth of colour models, many more than only RGB and XYZ. Both RGB and XYZ are models oriented on hardware and physics. They are not very intuitive for artistic use. In the introduction, we saw that artists often describe colours on the basis of attributes such as hue or lightness. But given a certain yellow it is not obvious which of R, G or B needs be adjusted to make it lighter. Models that express such visual concepts more directly would be preferable.

HSV and HLS model

Two common examples of such models order colours according to **H**ue, **S**aturation and **V**alue (=luminance) and **H**ue, **L**ightness and **S**aturation respectively. They are commonly referred to as HSV and HLS models (there is room for confusion between the abbreviation HSV and the abbreviation of the human visual system, HVS. In this section, I will therefore always spell out human visual system).

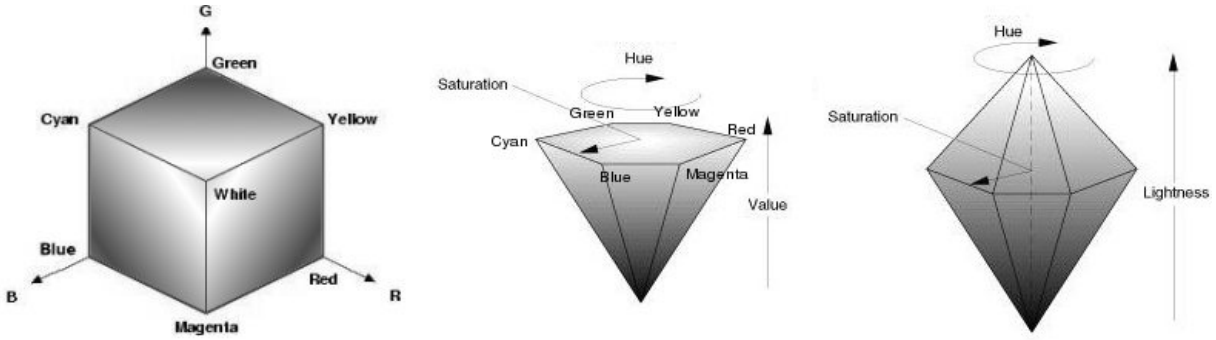


Figure 2.2 Visualisations of RGB, HSV, and HLS space (from left to right)

The models are best understood from their visualisations. They can all be derived from the RGB model. The RGB model is visualised as a cubic three dimensional space with one dimension for each red, green and blue. The coordinates refer to the varying mixtures of the three base colours.
HSV space can be visualised as a hexagonal cone. The base of the cone is obtained from the RGB cube by looking at the cube from the corner that represents white (as depicted in figure 2.2) and flattening it onto two dimensions. Hue is measured around the perimeter of the cone. Saturation is the distance from the centre and value decreases along the centre line towards the apex. Figure 3 in the appendix shows a coloured picture of the HSV space. HLS is then derived from HSV by grabbing white at the HSV base and pulling it up to form a double hexcone.

colour wheels

A common artistic tool based on the HSV or HLS colour model is the colour wheel. The colour wheel orders colours on a circle based on hue and saturation. It is approximately a slice of one of the hexcones with the edges smoothened into a circle. Again, colours are ordered such that hue changes with angle and saturation with the distance from the centre. Hue and saturation are essentially the polar coordinates of a point on the wheel. Depending on whether the wheel is derived from the HSV or HLS model, either lightness or value are assumed to be constant across the wheel. An important feature of the colour wheel is that complementary colours, colours that "cancel" each other to become white, are at opposite positions. This makes it useful as a tool to remove colour casts from images as we shall see in section 2.6. Figure 4 in the appendix shows an HLS colour wheel.

Flaws of wheels, HLS & HSV

Colour wheels, HSV and HLS, however, are seriously flawed with respect to the physical reality of the human visual system. The regular shape of the wheel suggests that the distance between minimum and maximum saturation was the same for every hue. It is not. There is no standardised conversion to obtain HSV or HLS coordinates from RGB. But most formulae, such as those provided by Foley, van Dam et. al [6] are based on the assumption that the luminance of a pixel was just the average of its RGB intensity values (R+G+B)/3. The human visual system, however, perceives some wavelengths as brighter than others which in reality leads to a weighted sum of about 0.3R+0.6G+0.1B with the exact coefficients depending on the exact primaries. HSV and HLS are still widely used in graphics applications, but are not recommended for reliable colour calculations (see for example question 36 in [4]). They are still useful, however, as a basis for interface elements and as means of simple visualisation. It would be more correct to call them colour ordering systems instead of colour models. They provide an easy and accessible way to find colours and to move between them, but do not offer the means of precise calculation and prediction a physical model does. Calculations based on these models will mostly be rough approximations only.

However, other intuitive models - and these are models in the physical sense - were developed

by the CIE. Instead of abstract lightsources X,Y and Z, they use accessible concepts like lightness. L*a*b* and L*u*v* are based on XYZ and allow direct manipulation of concepts such as lightness (L*) or hue (for example via a*b*, where a representstransition from red to green and and b* from yellow to blue) *L*a*b* & L*u*v**

L*a*b* and L*u*v* are useful models for colour calculations, a sound foundation to base an algorithm on that will generally give better results than HSV or HLS in chapter 5.

It should have become clear from the above that there is a wealth of models and ordering systems that all have different advantages and drawbacks. In the end, they are all only handles onto the mystery that colour still is. The difficulty is to know which one to use to achieve the desired effect.

2.6 Colour Correction

Colour Correction is altogether different from Colour Management. Colour Management is concerned with reproducing colours correctly according to a given specification. It is a rather technical endeavour, something that ideally would be taken care of by the system itself without any user intervention at all.

Colour correction (another commonly used term is colour grading), however, is a user feature. Colour correction in film is the artistic process of giving an image a certain appearance, a look, a feeling. The goal can be to remove unwanted colour casts on images (thus the term "correction") or to deliberately provoke them to create a certain atmosphere, such as warm evening light.

Colour Correction is best understood by examples: Assume a scene was partly shot on location, in front of a magnificent mountain landscape, and partly on set, as the camera swings around and shows an old medieval village that could not be built on location due to it being a natural reserve. Shooting on location takes a whole day, from the early morning until the afternoon. The final edited version will have to create the impression of one scene taking place during five consecutive minutes in one fixed location. *examples*

The artificial light on set contains more blue than daylight and appears colder. Afternoon light contains more orange and is warmer than light at midday. The brain is very good at evening out these differences, at keeping us from noting them (a phenomena called colour constancy), but the camera will pick them up and make them visible. If they are not evened out, the light changes will be noticeable to the viewer. Even subtle shifts will make the editing appear surprisingly uneven and the flow of the scene will be broken. The viewer will be very aware of the filming and the film will have failed to create the illusion it wanted.

Obviously a lot of effort is put into matching lighting conditions and cameras, but the final touch in matching the different parts of the scene often has to be done in colour correction.

Another different example: A film needs to be shot in a restricted environment. Access can only be granted for short times during the day. The scene, however, is supposed to take place in the evening. The solution is to shoot it at day and attempt to create a colour impression of evening twilight by using colour correction.

The colourist and director might even want to go beyond reality: To symbolise a characters distorted vision of his surroundings, they could manipulate the colours in strange ways. They might want to make a film appear like an old technicolor film. Or they might want scenes in an epic to appear as if they were immense wall painting (a prominent example is the Lord of the

Rings, in particular the last part). All these are applications for colour correction, a highly artistic task done by colourists.

Colour correction tools

The bottom line of all colour correction is the definition of a transfer function. The input are the original colours of the image, the output are the manipulated colours. To change brightness, for example, one could use a transfer function that simply adds a little intensity to each pixel such as $b_{new}(x)=b_{old}(x)+5$. The application could just prompt the user for a transfer function, apply it and be done. In a way, this is what the most powerful tool for colour correction, the curves tool, does: It allows the visual manipulation of a transfer curve that is then applied to the image. It does require a thorough understanding of colours, though, to achieve the desired effects.
Other tools are less flexible, but more intuitive. In a way, they hardcode a certain transfer function and let the user specify the input parameters. Colour wheels are an example. They allow changing the colour balance of an image to remove or deliberately create colour casts. The colour wheel is useful to predict how a change in one color component affects other colors. If the colourist detects a colour cast in an image, he finds the colour on the colour wheel and pushes the balance towards the colour that is opposite on the wheel. Since opposite colours are complementary, this will balance the cast.

There are many different mathematical models to describe colours and to allow operations on them such as changing their lightness or mixing them. The quality of the same colour correction tool can greatly vary with the underlying abstractions it uses in its implementation to achieve the desired effect. For instance, there is more than one way of manipulating lightness, but some ways are more in accordance with the nature of the HVS and will lead to better results.

Look Management

With an increasing number of films being edited digitally, the possibilities of artistic expression are increasing, too. A very recent concept is *look management*. After a colourist has adjusted one individual frame of a scene or a film, he stores his manipulations in a *look profile*. He can then view the whole film through this profile without changing the actual image data. He can create different looks and experiment with them before actually committing them to the image. Look profiles are a technique studios already use "manually", on the basis of their own tools or scripts. They essentially compare the original and the manipulated image colour by colour and create colour conversion tables to relate the original to the new colour values. These can then be applied to other images.

A common mistake that people make when trying to design something completely foolproof is to underestimate the ingenuity of complete fools.

Douglas Adams, English Novelist

3 Design

Colour management and colour correction in CinePaint are based on a couple of fundamental components such as formats and libraries. This section will explain their choice and briefly describe them.

Of course, one of the major "components" that this project was built upon is CinePaint itself. This required a substantial amount of familiarisation with the existing code base of CinePaint which a couple of lines will be spent on. Firstly, to point out some details important to the implementation of this project and secondly because the process resulted in another feature addition to CinePaint that is not directly linked to colour management or colour correction (though still to do with colour handling), but nonetheless interesting and worth mentioning.

3.1 Fundamental design decisions

Some of the initial design decisions that usually have to be taken in the process of developing a software application were predetermined by CinePaint, by the fact that this project is set within an already existing project. CinePaint uses C as programming language (although a gradual transition to C++ is planned) and gtk-1.2 as GUI toolkit.

Other decisions had to be taken by this project. These were:

- a storage format for colour space conversions as described in the previous chapters. This format will be the ICC profile standard
- a library to read the ICC profile format which will be litteCMS.
- a way to load and manage bigger numbers of profiles efficiently. This will be implemented using a simple caching mechanism based on hash tables and reference counters
- a storage format for look profiles which will be abstract ICC profiles.

3.1.1 A format for colour management conversions: ICC profiles

Justification of the decision

The first decision that had to be taken was how to store the colourspace conversions needed for colour management. It was somewhat predetermined by the fact that there really only is one standard format used for this purpose that is adopted by virtually all colour managed applications from *Apple ColorSync* to *Adobe Photoshop*. It is a file format based on a standard by the International Colour Consortium (ICC). Some applications use proprietary colour space transformations that are hard-coded into the application, but there is no other open profile standard available. The flexibility profiles offer over hard-coding and the acceptance without exception of the standard, make supporting the standard almost compulsory for any colour management system.

Also, there is an increasing number of applications that allow generation of ICC profiles tailored to a given working environment. This is important, because colours do not only depend on the model of a monitor, but also on factors like ambient light, the graphics adapter or the ink cartridges used in the printer. Profiling applications therefore often use specialised hardware, so-called colorimeters, to read the spectra of test colour patches that are output on the printer or the screen that is profiled. The profiles are then generated on the basis of the data from these devices. Unfortunately, support is still limited under Linux, but with an increasing number of applications supporting colour management this will hopefully change soon.

The ICC format

ICC profiles relate colour data in one colour space to data in another colour space by specifying a conversion function. The conversion can be expressed as a matrix, as a multi-dimensional look up table (LUT) or as a combination of both. Which approach is chosen depends on the complexity of the conversion. Converting between RGB and XYZ can be expressed in a 3x3 matrix, because XYZ values are merely linear combinations of RGB values. Conversion is carried out by multiplying the (R,G,B) triple with the matrix. The conversion from XYZ to CMYK (a printer's colour space), however, is highly non-linear. It can only be captured adequately in a big LUT. The LUT has one dimension per channel of the input colour space (e.g. 3 in case of RGB). A conversion is carried out by looking up the table cell that corresponds to the input channel values. The cell contains the corresponding colour in the output space. Usually only a subset of all possible input values is stored in the table and interpolation is used as appropriate.

PCSs

For every profile, either the source colourspace or the destination colourspace has to be CIE XYZ or CIE Lab in order to be able to flexibly connect profiles to conversion chains: to convert between two colour specifications, e.g. between the RGB of a scanner and the RGB of a monitor, two profiles are necessary. The profile of the scanner converts from scanner RGB to XYZ (or Lab), the profile of the monitor takes the data from there and converts from XYZ to monitor RGB. XYZ and Lab work as intermediate spaces to connect the profiles which is why in the context of the ICC standard they are termed *profile connection spaces* (PCS). The use of a PCS allows a maximum of flexibility in combining different profiles.

The profile format is complemented by a header containing meta data such as a description or the definition of a profile class. The most common profile classes are *input profiles* which are used for scanners or digital cameras, *display profiles* used for display devices such as monitors and *output profiles* for devices like printers.

device-link profiles

There is one other class of profiles that requires special mentioning: *device-link profiles*. They are equal in structure to the profiles described before, but do not translate from or to a PCS. Instead, they translate directly between two device colour spaces. In the previous example a device-link profile would convert straight from the scanner's colourspace to the monitor's colourspace without the intermediate step of converting to XYZ. Device-link profiles are created by melting two ordinary profiles into one which means multiplying their conversion matrices or combining their LUTs. Device-links are created for efficiency. If for example a display has to be refreshed often, one table look up can be considerably less expensive than two. The drawback is obviously that device-link profiles are tailored to one specific conversion and therefore less flexible.

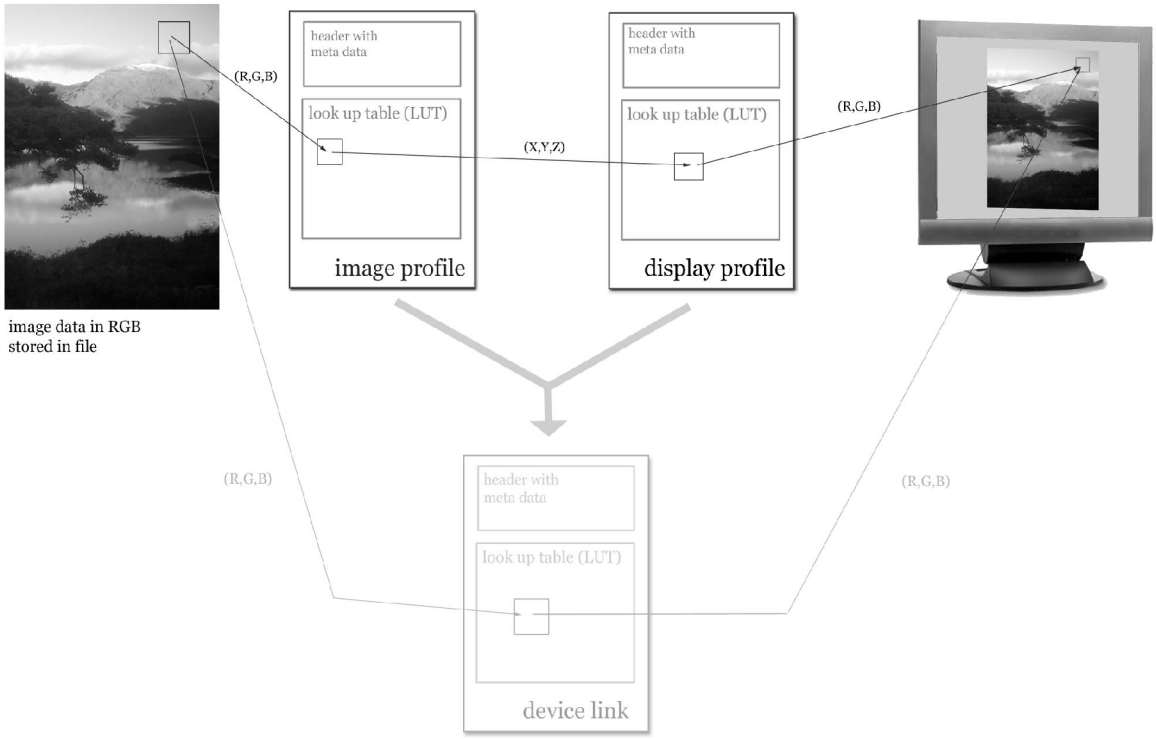


Figure 3.1 Colour management conversions using profiles

Figure 3.1 illustrates the conversion from an image's colour space to a monitor's colourspace and the possibility of making it more efficient using a device-link profile.

3.1.2 A colour management module: littleCMS

Having chosen to incorporate the ICC profile standard, the next step was to find a library to read profiles and to apply them to image data. In the standard architecture of a colour management system, such a library is commonly referred to as a *colour management module*.

CMS architecture

The architecture of a colour management system commonly consists of three components:a colour management module, a set of profiles and the colour management system (CMS) itself. The profiles represent the environment-depending configuration of the system. They customise the system for the devices in use such as printers or monitors. The CMM is the engine that drives the conversions. It is an arithmetic unit that reads the profiles and uses their look up tables and matrices to convert the input data. The colour management system is the enclosing unit that ties everything together and makes it accessible to the user. It manages the profiles, requests the CMM to perform transformations as necessary and lets the user set default options. Figure 3.2 shows these relations. The idea behind separating the CMM is to make it plug-replaceable. For Apple's ColorSync CMS, for instance, there is a variety of CMMs from different manufacturers such as Kodak or Apple itself.

Justification of the decision

For Linux there are three CMMs available or under development: *littleCMS* www.littlecms.com), *ArgyllCMS* (web.access.net.au/argyll/argyllcms.html) and *gCMS* (gcms.sourceforge.net). In spite of their names, they all are CMMs in the architecture outlined above. LittleCMS was chosen for this project for several reasons: its developer, Marti Marias, provides good and on-going support. It has a well-defined and stable interface which would in principle allow to later replace littleCMS with another CMM implementing the same interface. It is being used by an increasing number of prominent software packages on Linux such as *Scribus* (a desktop publishing package, www.scribus.net), *The GIMP* (www.gimp.org), *ImageMagick* (www.imagemagick.com) and *Qt* (the KDE interface toolkit, www.trolltech.com/product/qt). It has the best prospects of developing into a standard for Linux. In contrast to *ArgyllCMS* and *gCMS,* it is also available on Windows and MacOS which

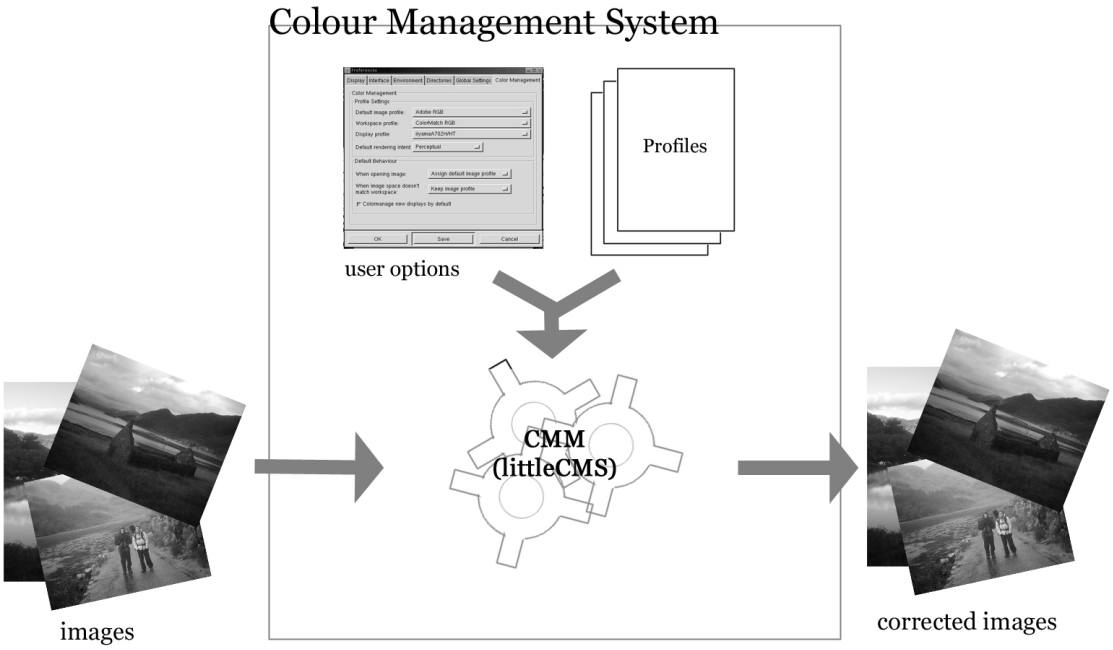


Figure 3.2 Architecture of a Colour Management System

is important for CinePaint which supports all three platforms. Last but not least, it is highly optimised and very small with a footprint of only 100K.

littleCMS

The core of littleCMS consists of functions to open profiles, to create profiles and to apply profiles to a buffer of image data. To apply a profile, it has to be turned into what littleCMS calls a transformation which is essentially an ordered list of profiles. If this transformation is then applied to image data, profile by profile is taken from the list and applied to it. If, for instance, we wanted to convert image data from scanner RGB to monitor RGB, we would first create a transformation consisting of the scanner and monitor profile and then apply this transformation to the data.

LittleCMS can make transformations more efficient by first melting the profiles in the transformation into a single profile, into what was previously described as a device-link. That way only one computation per colour is necessary, not one per colour and profile.

3.1.3 Handling many profiles: Storage and Caching

Even though littleCMS as the CMM provides functionality for reading and applying profiles, it is the colour management system's task to manage the profiles and to do so efficiently. They will have to be stored in a place where CinePaint can find them. Since many of them contain big look up tables that occupy considerable amounts of memory, access will have to be made efficient by avoiding duplication when several images use the same profile.

a storage directory

Unlike Windows and MacOS, Linux does not (yet) have built-in colour management functionality. Therefore there is no standardised directory to hold colour management profiles in the same way as there are, for example, directories for fonts. After consultation of other CMS related projects such as littleCMS and Scribus, it was decided to try and set an example hoping that others will go along with it and turn it into the standard Linux profile directory: CinePaint's profile directories were set to /usr/share/color/icc for system-wide profiles and ~/.color/icc for user profiles, following a suggestion by Marti Marias, the developer of littleCMS. These directories are scanned for profiles and the findings are presented whenever the user has to select a colour management profile.

profile cache

To make opening many profiles efficient, the implementation will have to include a caching mechanism which will be based on hashtables and reference counters. The mechanism ensures that no profile is opened more than once. Transformations will always be pre-computed in the form of device-links. Again there will be no duplication. Since pre-computation is time intense (about 2 seconds on a PII-300), computed transformations will be cached in temporary files on disc, if they are evicted from memory.

The mechanism will be described in more detail in chapter 4.3.1.

3.1.4 A format for look profiles: ICC abstract profiles

The question of which format to use arose again in the design of look profiles. As explained in the previous two chapters, look profiles capture the colour corrections done to an image in order to preview them on other images or to easily apply them to other images. There are two principal ways of achieving this.

choices

The corrections could be captured in a LUT relating the original pixel colours to the new pixel colours. This could work in just the same way it works for colour management profiles as described in section 3.1.1.

The other option would be to store the inputs of the tools used to achieve the correction, for example the settings of the colour wheels. These inputs could then be used to reapply the same

computations to other images.

It seems obvious that the first method is likely to turn out faster, the latter to be more precise. Ideally there would - and in case of CinePaint it is likely that there will be in the future - both options at the discretion of the user. Due to the limited amount of time for this project and the fact that look profiles are only one part of it, this project adopts the first approach. It is likely to be the more useful of the two. Considering that look profiles will often be applied "on-the-fly" to preview a scene through a given profile, efficiency is important. Besides, it ties in nicely with the first part of the project on colour management.

abstract profiles

As it turns out, the ICC profile standard already provides for this type of profiles. There is a class of profiles called *abstract profiles*. In structure, they are no different from ordinary profiles. Their function, however, is to apply colour manipulations to an image instead of converting between different colour representations. They usually use data in a PCS as both input and output and manipulate it. A profile could for example increase the image brightness by increasing all input colour values. LitteCMS provides the API to fill the LUT of these profiles with sample data.

3.2 Aside:familiarisation with the code base-binary fixed point

The familiarisation with the codebase took up a considerable amount of time during the first weeks of this project. The codebase inherited from The GIMP comes with hardly any documentation, not even the central data structures and building blocks are documented. So the code itself and information gathered on the developer's mailing list were all there was to work from.

Since code is not as readable as a novel, it seemed a good idea to combine the familiarisation with an initial implementational task that was suggested by Robin Rowe: To implement binary fixed point as a data format. What follows is a short explanation.

data formats

Colour data in CinePaint is stored as RGB triplets in four possible formats: 8-bit unsigned integer, with (0,0,0) representing black and (255,255,255) representing white, 16-bit unsigned integer between (0,0,0) and (65535,65535,65535) and 16-bit and 32-bit float between (0.0,0.0,0.0) and (1.0,1.0,1.0). Today's displays cannot diplay colour of more than 8-bit depth (256 different settings for red, green and blue), but film scanners can often read data in higher resolution. For many image manipulations the result is better, if the data is stored and manipulated in a higher bit depth with more detail and only downcast when it has to be displayed.

whiter than white

Along the same lines, it can be advantageous to be able to store values that are whiter than white. White is always relative. It is whatever the display device or the paper define it to be. Paper often appears white until it is compared to another sheet that is even "whiter". White on a monitor is (255,255,255). Values higher than this would simply appear white, too. However, there are reasons to store these values, anyway. An example may be a picture that contains a lot of bright values. Fires and explosions are common examples in film. Increasing the brightness of such a picture will result in some values going beyond the white limit. If this limit is enforced, values beyond it are clipped and simply set to the maximum value. If there is a possibility to store over-white, such values will store be stored as they are (for example at 1.2), even though on the screen they will appear white all the same.

The advantage becomes obvious when the brightness is decreased again. In the first case, all over-white values were lost. They have all become ordinary white. Reducing their brightness will turn them now into an ugly grey without any detail. In the latter case, the values have been

preserved and as they slip below the white limit again, the detail reappears on the screen. Another use case is scanning film. Film captures subtle differences in very bright colours that on a monitor would all appear as white. So when scanning, the white limit is set to the monitor's white, but values beyond it are stored, anyway. If brightness is later decreased , detail will appear that was always there on the film, just could not be seen on the screen.

BFP

To be able to store over-white values requires a suitable data format. Unsigned integer is not suitable, since white is already defined to be the maximum value. There is no 8-bit integer value beyond 255. float could be used, but is in practise currently clipped to 1.0 in CinePaint. Therefore 16-bit binary fixed point (BFP) was introduced as the first format that can store over-white. In 16-bit BFP the highest order bit is interpreted as the single digit before the decimal point (0 or 1) and the remaining 15 bits as the decimal places. Thus BFP ranges from 0.0 to 1.99 and can store over-white.

To represent over-white on the screen where it normally cannot be seen required allowing the user to choose a strong warning colour that is used whenever colours are beyond white.

This first task gave a good overview of the core code and produced an encouraging first result.

3.3 CinePaint internals

A few short remarks about the CinePaint structure should be made to aid the further understanding of the implementation:

The different data formats have already been mentioned. Unfortunately at the moment they are not implemented very efficiently. The introduction of a new format requires rewriting almost all functions that manipulate image data. The part of this project concerned with colour correction will therefore focus on 8-bit unsigned integer as a format and the extension to other formats will be left for the future.

Image data in CinePaint is currently always stored as RGB data, or optionally as RGBA where A, the alpha channel, stores the opacity of a pixel.

GImage & GDisplay

Two of the most important data structures of CinePaint with respect to this project are GImage and GDisplay. GImage represents a physical image. It contains its dimensions, image data and other information. GDisplay is a view onto the image displayed on screen. Several GDisplays can refer to the same GImage. For example, the user might want to display the same image twice at different zoom levels or with different display filters applied. The physical image behind the two views, however, is the same and manipulations of the image in one display will automatically affect the other.

flipbook

CinePaint provides a feature called the flipbook which has significance to the implementation of look profiles described in chapter 5. The flipbook allows you to concurrently load a sequence of numbered images that are thought of as the frames of film, to move through them and to play them like a film sequence.

If a sufficient number of management layers are superimposed on top of each other, it can be assured that disaster is not left to chance.

Norman Augustine, US Author

Implementation I: Colour Management

This chapter and the following one describe the work done in this project in detail. Whilst this chapter deals with colour management, the next one will cover colour correction. Both chapters follow the same structure. They start out by pointing out the major challenges that were faced in the implementation of this project. They then describe the implementation from a user's point of view, much in the style of a user's manual. They conclude by describing the implementational details.

4.1 Challenges

The three central issues to resolve in the implementation of colour management were efficient handling of profiles, fault tolerance and an accessible and understandable user interface.

efficiency

Efficiency probably was the single most important issue. This concerns efficiency both in terms of runtime and in terms of memory consumption. It is particularly important, because colour transformations take place "on-the-fly": the actual image data stays untouched, only as it is sent to the display it is transformed.

There is a variety of factors that can cause performance issues:

- Profiles are based on big multi-dimensional look up tables (LUTs) that can take up significant amounts of main memory. Sizes range from 200K up to megabytes each, depending on the number of sample points stored in the table and the number of colour channels in the image.
- Merging the profiles involved in a transformation into a device-link is a time intense task.
- Finally, transforming big amounts of image data in itself is an additional performance burden.

fault tolerance

Fault tolerance is important, because colour management relies on external components, namely the profile files. What happens, if between two runs of the program the needed profiles have been moved or deleted, for example by a diligent sysadmin who could not make sense of them? CinePaint has to be able to deal with this by using fail-safe defaults.

user interface

And finally, the user interface had to provide enough flexibility to provide for a lot of different environments and needs. Some users may want to use the same profile for all their images. Other users need different profiles and will have to decide on an image by image basis. The images provided may not be available in the colourspace they are needed in which makes conversion necessary.
The difficulty was to define a workflow that was flexible enough without defining an overwhelming and confusing amount of options.

4.2 The User's View

4.2.1 Configuration

storage path

To be able to use colour management, you have to configure it for your environment. The most important step is providing and selecting the necessary ICC profiles. Profiles are selected on the colour management tab of CinePaint's preference dialog which is located in the menu File>Preferences....(see figure 4.1) By default, profiles are searched for in the directories /usr/share/color/icc and ~/.color/icc. The idea is that the first path is used for system-wide profiles and the second for user-specific ones. You can change these settings on the directories tab of the preference dialog.

On the colour management tab there are three profile selections to be made: a *default image profile*, a *workspace profile* and a *display profile*. This structure has become a de facto standard or at least good practice in standard colour management workflows and related applications such as Adobe Photoshop or Apple ColorSync. The selections for default image and workspace profile will contain all profiles that were found in the profile directories. The selection for display profile will only show profiles of class display profile.

To correctly display an image, CinePaint essentially needs to know two profiles: the profile that

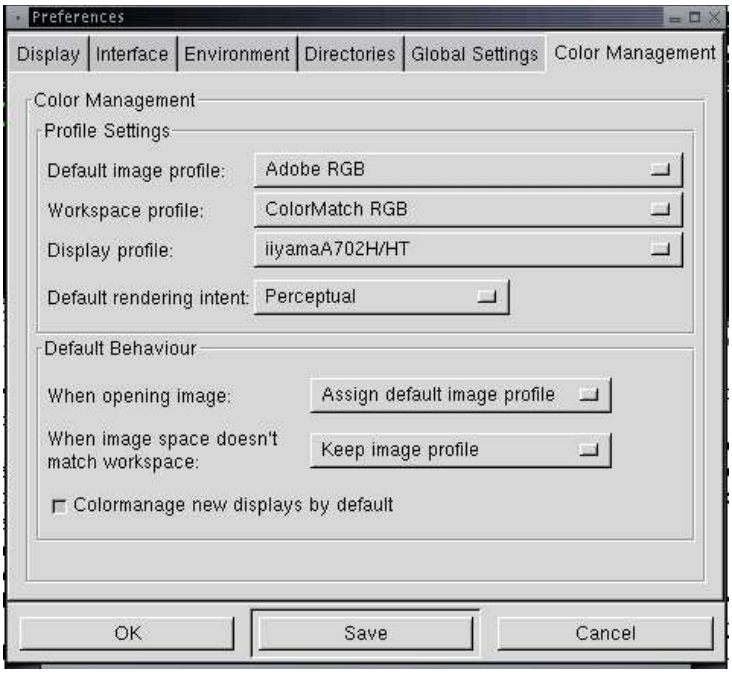


Figure 4.1 Colour Management Preferences

describes the image data and the profile that describes the display device.

display profile

The display profile is selected directly via the setting of the same name. The display profile describes the display device you are using such as your monitor and is a system-wide setting. It will be used in displaying all colour-managed images. CinePaint currently does not provide for systems using two different display devices at the same time.

image profile

The image profile is assigned to the image when it is opened. In the preference dialog you can determine how this is done. You can choose to always assign a selected default image profile, to never assign a profile or you can choose to be prompted for the profile to be assigned whenever you open an image. You can see which profile is currently assigned to an image from the description in the image's title bar.

embedded profiles

In principle there is another way in which a profile can be assigned to an image: by embedding it into the image file itself. Several file formats such as TIFF and JPEG allow embedding of profiles. However, detection of embedded profiles is not provided for in the current implementation. This is because they have no high priority in movie studio applications. An embedded profile can increase the file size noticeably. A digitised movie consists of hundreds of thousands of individual files, one per frame of the movie. They all use the same profile, since they were scanned using the same scanner. It would be an incredible waste of space to embed a profile into every single one of them. Instead you would keep one single profile in a separate file and define it as the default image profile. Embedded profiles are useful in pre-press applications to exchange individual images with a printer.

workspace profile

Having explained the display profile and the default image profile, this still leaves the third profile setting in the preferences. The workspace profile defines your preferred colourspace, your in-house colourspace. Most professional users such as newspapers, agencies or movie studios will do all their work in one fixed colourspace because standardisation greatly helps coherent workflows and avoids confusion.

default behaviour

The workspace profile is the profile that CinePaint applies to every newly created image and that it may enforce for opened image if you choose so. As explained above, CinePaint assigns a profile to a newly opened image according to the rule selected in the user preferences. If this profile does not match the workspace profile, CinePaint can either ignore the mismatch and keep the image profile, automatically convert the image to the workspace or consult the user on whether to convert or not. Which action is taken depends on the preference setting *When image space doesn't match workspace*.

rendering intent

Finally, you can define a default rendering intent to use colour management. As explained in chapter 2.4, the rendering intent defines the way colours are approximated that are not in the display device's gamut. For a detailed description of the options available refer to any introductory text on colour management such as [10] or [11] The intent selected here is the default option. You will later be able to change the rendering intent for each display individually.

A final note: For each of the profiles settings it is possible to select no profile at all. Also, you can choose to have no profile assigned to a newly opened image. Notice, however, that CinePaint will only be able to colourmanage an image, if you have set a display profile in the preferences and have assigned a profile to the image. Should you forget to do so, CinePaint will remind you.

4.2.2 Use

Now that you are all set up, you can start to have your images colourmanaged. Most of the colourmanagement functionality unfolds in the background without the need for any direct interaction - just as it should: As the user, you can get on with your work and rely on the colours to be correct without having to think about it.

Since colourmanagement causes a certain loss of performance, it is at your discretion whether to use it or not. Notice the toggle entry View>Colormanage Display in each display's menu. It allows you to switch colourmanagement on or off for each display individually.

You can change the rendering intent used by CinePaint for a particular display using the menu View>Rendering Intent. By default this is set to whatever you selected in the preferences.

assigning and converting

The title bar of each display shows you which profile is currently assigned to an image. If you want to assign a different profile, you can do so by using the menu Image>Assign ICC Profile... You will be prompted for the new profile to be assigned. If you choose to assign no profile, colourmanagement will be switched off for all displays of this image. (Remember that CinePaint always needs to know an image and a display profile to be able to colourmanage)

Finally, you can convert an image's data into another profile's colourspace. Obviously, this also requires a profile to be assigned to the image, so that CinePaint knows where to convert from.

In contrast to ordinary colourmanagement which does not affect the image data, this will actually manipulate it to transfer it to the new colourspace. Again, you are prompted for a profile to convert to and in this case also for a rendering intent to use in the conversion.

4.3 Behind the scenes

When colour management is switched on, the standard control flow as explained in chapter 2 applies: The image data is sent through the profile assigned to the image and through the display profile. The image profile translates the data to the profile connection space (PCS) which is either CIE Lab or CIE XYZ, the display profile takes it from there to the display device's space.

Each GImage structure contains a pointer to the image's respective profile. The display profile is a global setting and stored in a global variable.

4.3.1 Efficiency I - The Profile cache

A lot of effort in the implementation went into providing an efficient profile handling. It is based on a two-layered caching mechanism using hash tables and reference counters. As a first layer, profiles are cached. No profile is opened more than once, if it is used by more than one display. The implementation provides a function to request a profile identified by its filename. Internally, profiles are identified by a combination of information from their header. This assures that equality between profiles is detected even if identical profiles are stored in different files or (in the future) are embedded inside image files. If the requested profile is found to be open already, it is returned from the cache. A reference counter keeps track of how often a profile is in use. The counterpart is a function to return the profile to the CMS. The reference counter is decreased and, as it reaches zero, the profile is closed.

Transformation cache

On top of the profile cache, transformations are cached. Again, the implementation provides a function to retrieve a transformation and a function to return it. The function to request a transformation is essentially a wrapper around littleCMS's transformation generation function. It compiles the parameters such as the profile handles into a hash key and checks the cache.

Transformations are somewhat different from profiles. Opening a profile takes no noticeable time, it is equivalent to opening a file. Profiles are usually kept by the requester for a long time. An image profile, for example, is kept during the whole lifetime of the GImage structure. This is not a problem, because profile handles are merely sophisticated file handles. The files themselves are only loaded into memory as needed.

Transformations, on the other hand, are often bigger and lie entirely in memory. To allow efficient cache handling, they are usually requested for one transformation and then returned immediately. Pre-computing transformations into a device-link is time intense. On the PII-300 that development was done on, it took very noticeable time (about 2 seconds).

Because of these differences, transformations need a slightly more sophisticated caching system than profiles. It should not only avoid keeping the same transformation open several times, but also unnecessary recalculation of device-links.

As a first measure, a copy of every calculated transformation is stored in a temporary file on disc. If ever it is evicted from memory, it can be reloaded quickly and without the need to recompute it. In addition, because transformations are usually requested for short times only, but often for several subsequent times, the last used profiles are kept in memory even when the reference counter has reached zero. How many profiles are kept is a compile time option. If all places are taken, transformations are evicted on a least recently used basis.
This is simply implemented as a ordered list of transformation handles for which the reference count is zero. When the last reference to a transformation is returned, the transformation is added to the end of the list. If a transformation is requested again, it is removed from the list. If at any point the list is full, the transformation that comes first in it is evicted to make room.

Brevity and conciseness are the parents of correction.

Hosea Ballou, US clergyman

The standard usecase in movie editing is keeping many open images simultaneously, each of them using the same image profile, since they are frames of the same film, and the same display profile, since they are viewed on the same screen. In this situation the caching mechanism is particularly effective. Without it, each image would open its own copy of the profile and generate its own transformation, keeping all transformations in memory at the same time or heavily swapping between memory and disc. The caching mechanism ensures that the image profile is opened exactly once, that the transformation is calculated only once and that this transformation can be kept permanently in main memory for fast access.

4.3.2 Efficiency II - Display buffer

To make the actual transformation of image data more efficient, parts of the CinePaint rendering engine had to be restructured. The rendering engine was inherited from the GIMP and had grown in rather unfortunate and inconcise ways over the years. Since this project is not mainly concerned with this part of CinePaint, refactoring was limited to the necessary changes to allow the introduction of a display buffer for every display of an image that holds its final screen version.

Before, the rendering engine only used one buffer for each image, not one for each display of it. Besides, the use of the buffer was inconcise. For example, layer composition was done within the buffer, whilst channel decomposition (which happens if for example you switch off the red channel in an image) was done on-the-fly, as the image buffer was copied onto the display

The new display buffer, however, contains the final screen version of the image. A screen refresh thus only involves copying from the buffer to the screen without any further transformations apart from scaling to the zoom level. It is necessary to have a buffer for each display and not only for each image, because with the introduction of look profiles in the next chapter the data shown in each display can differ.

The new buffer structure permits to reduce the task of colourmanagement to a single transformation of the display buffer, such that afterwards the buffer contains the corrected data. The speed increase compared to transforming every time data is copied from the buffer onto the screen is enormous.
Introducing one buffer for each display increases memory consumption, but it is certainly made up for by the improved performance. Besides, opening more than two or three displays of the same image is a very uncommon usecase. After all, the space on the screen to view several displays at the same time quickly reaches its limits.

4.3.3 Fault Tolerance

Fault tolerance mainly refers to the CMS's resistance to stumbling over lacking or non-defined profiles. The user may delete or move profiles that were set as display or default image profile. He might decide to not assign a profile to an image. All this will make colourmanaging an image impossible.

Therefore the CMS examines the settings carefully each time a profile is needed. It provides an initialisation function that is called from CinePaint's main initialisation function that sanity

checks all settings and deletes settings that have become invalid (such as profiles that do not exist anymore).

The only hazard that CMS will stumble across is deleting an opened profile during the execution of CinePaint, because profiles are only loaded into memory when needed.. This was not considered a serious problem, however. After all, if a program is deleted during its execution, it will eventually crash, too.

Implementation II: Colour Correction

This chapter continues with the detailed description of the work done in this project. Whilst the previous chapter dealt with colour management, this one will cover colour correction. It follows the same structure. It starts out by pointing out the major challenges that were faced in the implementation of this project. It then describes the implementation from a user's point of view, much in the style of a user's manual. It concludes by describing the implementational details.

Note: In spite of CinePaint offering a range of different bit depths for storing image data, Colour Correction was only implemented for 8-bit. This is because the implementation of different bit depths in CinePaint is currently rather inefficient. Adding new bit depths requires a substantial amount of code duplication, of cutting and pasting and replacing pointer types which is neither a very challenging nor a very interesting task. However, colour correction will be readily extensible, especially if the CinePaint bit depth implementation is re-factored in the future as is hoped for.

5.1 Challenges

The central difficulties in implementing colour correction were clearly specifying the requirements and defining an underlying algorithm for the colour wheels.

gaining insight

The problem with specifying the requirements was to obtain enough insight into the area of application. Unfortunately there is little information available on the art of colour correction in film. Asked by Robin Rowe, the CinePaint project leader, about where to learn more about film colour correction, a colourist at *Flash Filmworks*, the studio that used CinePaint for *The Last Samurai*, answered: "Only in a training program or on the job experience at a film lab or color suite - as far as I am aware"

Colour correction is a skill that is based on a lot of experience and the advice one receives depends on the colourist asked and the kind of tools he used in the past: "*If you get this kind of a strange green cast in the blacks, it usually helps to tweak here a little. And adjusting this is always a good idea, too.*" Asking three colourists will likely result in five answers. And after all, there was no colourist designated to this project to answer questions. For obvious reasons, a trip to Hollywood to visit one of the studios CinePaint is used at was out of question.

I tried to close the gap in two ways: Firstly, by exploring packages available to me at the university. These were Adobe Photoshop, Adobe Premiere and Discreet Combustion. Adobe Photoshop is a still image retouching program and currently the standard that everyone has to bear comparison with, Adobe Premiere is a hobbyistic video editing tool and Discreet Combustion is already a considerable way up the professional scale towards film editing.
Secondly, I phoned the BBC to find out about the possibility of witnessing a colour grading session. Eventually, I was able to sit through a day's worth of colour correction using yet another package (Avid Symphony which is the industry standard in video editing) from which I took away a couple of valuable points.

interface

The user interface is one of the most crucial parts of designing colour correction tools. As the former Director of Software Development for Leitch Video (www.leitch.com) pointed out to me: *"The back end processing algorithms were never the issue. For the product to succeed, it had to intuitive to operate and fit in the job flow model used in video and film post. [...] Building an interface blind, it is hard to get the UI designed and laid out just right so it will work in a production environment."*

algorithm

Defining the underlying algorithm proved equally difficult. Few guidelines were available about what exactly is desired. There is specialised literature on colours some of which might have helped to some extent, but it is hard to access unless one is willing to buy it. Neither the university library nor the city library nor the art college had anything available. The central question to answer was which colour models to base the implementation on.

5.2 The user's view

the wheels

The colour correction tool is accessible via the menu Image>Colors>Color Correction (see figure 5.1). Its main feature are three colour wheels that allow intuitive manipulation of the colour balance of an image. Manipulation is done individually for shadows (the wheel marked "Sh"), midtones ("Mid") and highlights ("Hi") in the image. The terms refer to the brightness range of the pixels that are most affected by the transformation. A division into three ranges is standard in graphics applications, since it is often found that certain problems only appear in certain parts of the image, such as a black suit exhibiting a slightly green colour cast.

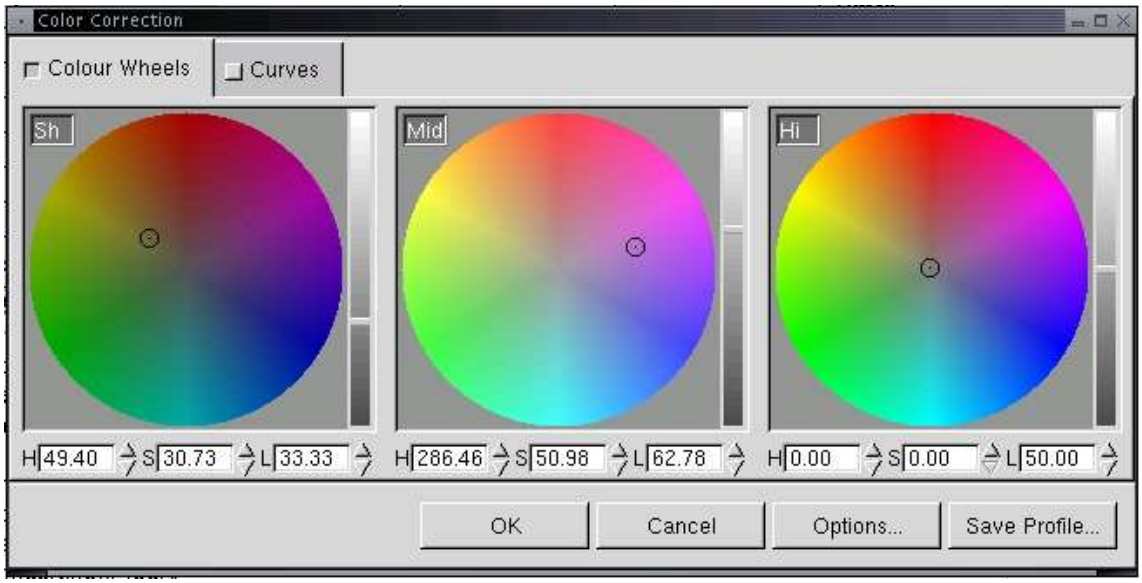


Figure 5.1 The colour wheels colour corrector

The wheel display is based on the HLS colour model which was briefly explained in chapter 2.5. The wheel you see is a slice through the HLS double cone. The slider on the right of each wheel allows you to change the lightness moving up and down the cone. Using the handle inside the wheel you can change the colour balance of the image. Dragging the handle in any one direction will tilt the colour balance in that given direction. The further you move it out of the centre, the stronger, the more saturated, the colour cast will become.

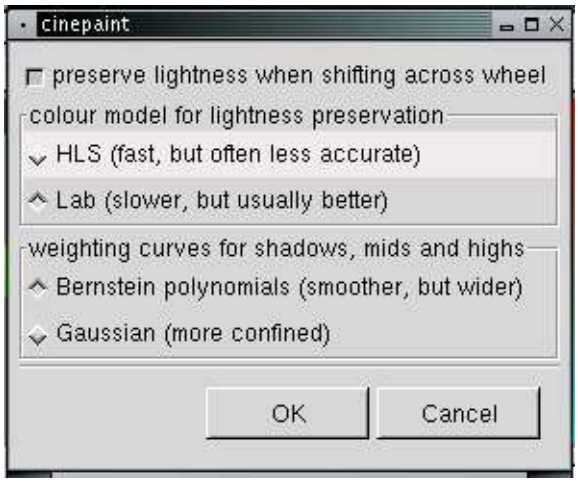
The wheels allow easy and very intuitive adjustment of the image's colour balance. If you note a colour cast in the image, the right step to take is shifting the corresponding wheel handle in the opposite direction to balance it. If, for example, the image exhibits a yellow colour cast, you shift the handle away from yellow towards blue.

To allow exact adjustment, you can enter values for **H**ue, **S**aturation and **L**ightness into the three boxes underneath the wheel. Lightness and saturation range between 0.00 and 100.0. Hue stretches from 0.00 to 360.00.

Note that you can also resize the dialog which will result in the wheels growing to give you more precision in using the wheel handle, if you need it.

The effect of each of the wheels can be switched on and off individually using the toggle button in the top left corner of each wheel which is labeled "Sh", "Mid" and "Hi" respectively. The overall effect of all wheels can be enabled and disabled using the toggle button on the *Colour Wheels* tab. You will notice that there is a *Curves* tab, too. This is still without function. It merely indicates how colour wheels and curves should be integrated in the future to create a single power colour correction tool.

Clicking the *Options...* button will open the options dialog (see figure 5.2) which allows you to fine tune the algorithm that is used to apply your colour changes. *options*
Changing the colour balance of an image is achieved by manipulating the intensities of individual colour components. This is likely to result in an overall change of the lightness of the image. Normally, the algorithm will attempt to counteract any lightness changes apart from those explicitly requested by moving the lightness slider. The first switch in the option dialog


cinepaint
preserve lightness when shifting across wheel
colour model for lightness preservation
HLS (fast, but often less accurate)
Lab (slower, but usually better)
weighting curves for shadows, mids and highs
Bernstein polynomials (smoother, but wider)
Gaussian (more confined)
OK
Cancel
Figure 5.2The Colour Correction Options Dialog

allows you to switch this correction off, resulting in a simple manipulation of the colour components irrespective of the caused lightness changes. See figure 6 in the appendix for an example.
If you choose to preserve lightness, the dialog allows you to choose the colour model that the calculations to preserve lightness are based on. HLS is simpler and faster, but often less accurate. You will notice that in many cases the colour cast tends to be too light. For an example, see figure 7 in the appendix. L*a*b* is slower, but the colorimetrically correct model. It even takes into consideration your particular image's colourspace if you have assigned an image profile for colour management.
As a last option, you can choose the weighting curves that are used to distinguish between highlights, midtones and shadows. A Gaussian will give you more confined ranges, but especially when adjusting more than one range at a time with the adjustments adding up the result sometimes turns out to light. Bernstein polynomials stretch the three ranges further, but the results are smoother. See figure 5 in the appendix for a comparison. More explanations on the weighting curves are given in the next section of this chapter.

saving look profiles

The button *Save Profile...* will allow you to store your manipulation in a look profile. You will be prompted to enter a file name and a number of sample points. The number of sample points is the number of colour samples taken per colour channel to fill the multidimensional look up table of the profile. The default of 13 will result in 13x13x13 samples to be taken. As you may assume, a higher number of samples leads to higher precision, but more expensive computation.

A look profile allows you to preview other images or whole sequences of images with a manipulation applied without actually changing the image data. You can create several profiles for a scene, watch the scene through each of them using the flipbook and finally decide which one you like best. You are also able to commit the manipulations to the image data, meaning permanently changing the data in accordance with the profile.

using look profiles

Look profiles are used through the look profile dialog (figure 5.3) in the menu View>Look Profiles... Each display owns it own dialog. The dialog shows you all profiles that were found in a given storage directory. You select the storage directory by clicking the *open*-icon. The dialog will remember the storage directory and the profile save dialog will use the same directory.

Figure 5.3 The Look Profiles dialog

The handling of profiles is similar to layers. To watch an image through a look profile, simply click onto the white space on the left of its name. The profile will be switched on, symbolised by a little pair of sunglasses. You can switch on several profiles at the same time.

To commit the current look profile settings to an image, click onto the second icon in the look profile dialog. The last icon lets you commit the profiles to all frames currently loaded into the flipbook. In either case, the image data will be manipulated and the profiles will be switched off, since after committing them, they are already part of the image data.

5.3 Behind the Scenes

5.3.1 Drawing the colour wheel

In contrast to all other interface elements like buttons and inputboxes that are already provided as elements of CinePaint's interface toolkit gtk, the colour wheels had to be implemented manually. The underlying gtk interface element is a drawing area providing simple drawing primitives. There is one drawing area for each wheel and each lightness slider. The slider is drawn by iterating over the pixel positions and translating them into RGB gray values between (0,0,0) and (255,255,255) (equal proportions of R,G and B give grey). The position on the slider is scaled to a range of [0,100] and determines the lightness.

The wheel is drawn by iterating over the pixels in its drawing area and calculating the appropriate RGB values based on the HLS colour model as defined by Foley, van Dam et. Al [6].

First, the wheel's centre coordinates and radius are determined from the width and height of the drawing area. Afterwards, coordinates can be translated such that the origin is at the wheel's centre. Hue and saturation then simply correspond to the polar coordinates θ and r, such that translating from a position in the circle to hue and saturation is equivalent to translating from

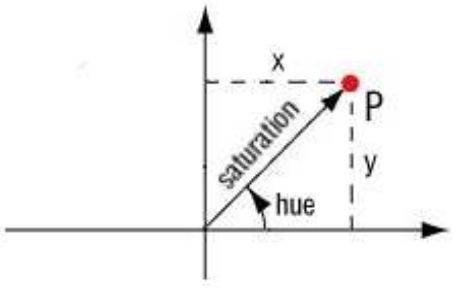


Figure 5.4 hue and saturation as polar coordinates

Cartesian to polar coordinates. Figure 5.4 illustrates this.

If the resulting value for r is greater than the radius, the pixel lies outside the circle and is ignored. Otherwise r is scaled to a range of [0,100] to give the saturation. θ directly determines the hue.

Given hue, saturation and the lightness determined by the lightness slider which is constant for all (x,y) on the wheel, we can convert the HLS values into monitor RGB values using Foley, van Dam et al's algorithm. The pixel is then coloured in the resulting colour.

To make the drawing more efficient and to avoid flickering, the wheel is obviously drawn onto an off-screen buffer that is copied onto the screen when a refresh is requested.

5.3.2 Colour correction

What the colour corrector essentially does is shifting the image's colour balance towards a given hue, manipulating the proportions of the image's red, green and blue channels. This means either adding the given hue or subtracting the complementary hue, the hue that "cancels" the selected hue to white. Adding or subtracting intensity on the image's channels may change the image's overall lightness, so the algorithm needs to provide means to maintain the lightness constant.

preserving lightness

There is no standard way of preserving lightness. Different approaches sometimes differ in the quality of the result, sometimes in the desired effect. It was explained before that it proved difficult to specify the exact effect required. Most likely there is no singular definition and no "best" approach and having different approaches to attempt for a given task may be advantageous. Therefore, the implementation provides two alternatives and leaves their use at the discretion of the user. One is based on the simple, but flawed HLS model to maintain lightness, the other one is based on the more complex L*a*b* model.

The algorithm is divided into two steps: Changing the colour balance and readjusting the channels to maintain lightness. Changing the colour balance of an image in the physical sense means changing the spectrum it radiates, adding or subtracting intensities of certain wavelengths. To model this, colour model which is linear with intensity is needed, a model that allows simple addition of intensities. This is the case for RGB and XYZ. RGB will be less expensive in terms of computation time, since data in CinePaint is already stored in RGB. If at a later point it was desired to do the same manipulation in other colour spaces such as CMYK, one could convert loss-free to XYZ using the colour management system, carry out the calculations in XYZ and finally convert back to the original space.

shifting the balance

Changing the colour balance towards a given hue then amounts to interpolating between the current colour of each pixel and the hue to shift towards. If a pixel has colour (120,140,55) and is to be shifted towards blue at (0,0,255), the two values are interpolated giving (60,70,100) which is considerably bluer than the original value.

The interpolation is weighted by the saturation selected on the wheel as a measure of how far to shift towards the given hue. The results above would have been for a saturation of 50%. A saturation of 25% would have resulted in (30,35,50). If the user decided to not keep lightness constant, this already is the new pixel colour.

Otherwise the algorithm proceeds with the second step of adjusting the lightness of the new colour. The goal is to assure that the lightness before and after the manipulation are the same. RGB does not allow easy manipulation of lightness. HLS and L*a*b*, however, do in the L and L* component respectively. So original pixel colour and the new pixel colour are converted to one of the two spaces, depending on the user setting. The lightness of the new colour is then set

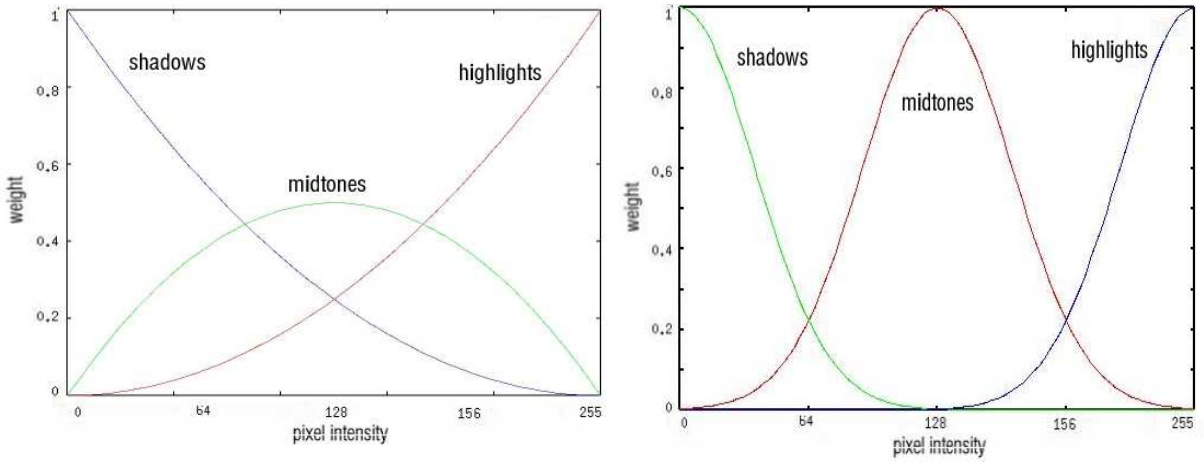


Figure 5.5 Lightness weighting curves: Bernstein polynomials (left) and Gaussian (right)

equal to the lightness of the old colour and the result is converted back to RGB.
If the HLS model is used, calculations are again based on Foley, van Dam et. al's algorithm. If the L*a*b* is used, conversions are carried out through littleCMS and the colour management system previously implemented in this project. If the user has specified a profile for the image, this will be taken into consideration to achieve colorimetrically optimal results. If no profile has been specified, a standard RGB profile will be assumed. I will elaborate further on the differences resulting from the use of HLS and L*a*b* respectively in the evaluation in chapter 6.3.

weight functions

This only leaves the interpretation of the brightness ranges of shadows, midtones and highlights to be specified. They are implemented as simple weighting functions applied to the manipulation. The weight ranges between 0.0 and 1.0 and is calculated in function of the original pixel intensity. For shadows, the weighting function returns a high weight for pixels of low brightness. For midtones the weight peaks in the middle. For highlights it increases with increasing brightness. Two alternative weighting functions were implemented and left for the user to choose: Bernstein polynomials and Gaussian bell curves. A Gaussian function gives more confined ranges, but does not necessarily add up well when manipulations of different ranges are combined. A Gaussian function can be computed effectively, because of its symmetry. It is sufficient to compute half the function and then reflect it. Bernstein polynomials have the nice property of adding up to one at any point which results in very smooth transitions between the different lightness ranges. Figure 5.5 depicts the two different functions.

The weight functions are precalculated for all 256 possible lightness values and stored in an array. To extend colour wheels to higher precisions, it will be necessary to interpolate or to experiment with direct calculation.

5.3.3 Look Profiles

applying profiles

Chapter 3.1.4 explained that look profiles can be implemented as abstract ICC profiles. The previous chapter on the implementation of colourmanagement explained that the image data is transformed through the image's profile and the display's profile before it is displayed. These two ideas can readily be combined. In the creation of transformations in littleCMS, the number of profiles is irrelevant. littleCMS receives an array of profile pointers and merges them into a single device-link transformation. Look profiles can therefore be simply combined with the remaining profiles. The transformation then starts with the image profile converting from the

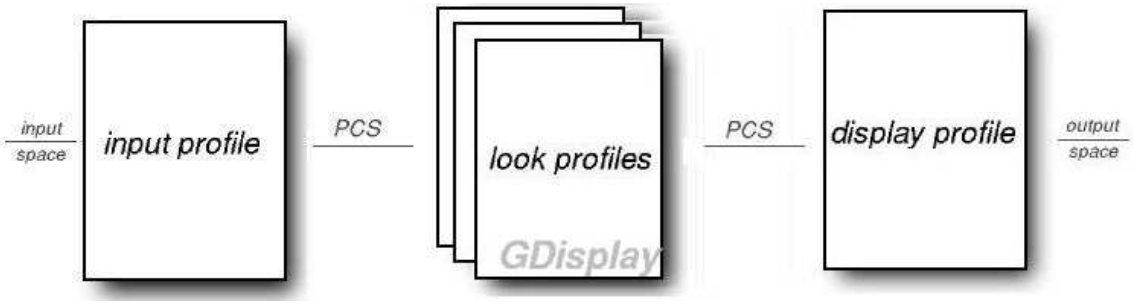


Figure 5.6 The complete profile chain

image space to a PCS. It continues through one or several look profiles that manipulate the data, but leave it in the PCS. It concludes with the display profile converting the data into the display's colourspace. Figure 5.6 illustrates this.

Look profiles are kept in singly-linked list structures that are assigned to each GDisplay. This makes it possible to have two displays showing the same image with different look profiles applied.

Look profiles blend seamlessly into the existing colour management framework - with one exception: look profiles should still work when colourmanagement is switched off for a display or even when no image or display profile is defined. In that case the image profile and the display profile in the chain of figure 5.6 are replaced by a standard RGB profile: As a result, data is transformed from RGB to the profile connection space, is manipulated in the PCS using the look profiles and then converted back into the original RGB space to be displayed. The display is in this case not corrected, since colour management is switched off, but the look profile will still be applied.

Committing profiles to an image uses exactly the same transformation, only that the result is not sent to the display, but stored back into the image.

creating profiles

Creating a look profile essentially means filling its multi-dimensional look up table (LUT) with appropriate values. Functionality for profile creation is provided by the littleCMS API. The creation starts by completing the information in the profile header, such as profile class and input and output colour space. Since look profiles will later be used between input profile and display profile (as depicted in figure 5.6), they have to use a profile connection space (PCS) as both input and output colour space.

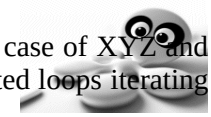
The LUT is filled by providing a callback function that is repeatedly invoked by littleCMS, once for each sample point in the LUT. The function receives input colour values from littleCMS and is expected to return the corresponding output colours.

Input and output colours are sampled from the image before and after the manipulation. As seen in chapter 3.1.1, the LUT will not contain all possible values, but interpolation will be used when the profile is later applied. Therefore it is important to provide evenly spaced samples during the creation of the profile. Based on the normal image data this is not necessarily possible. A picture of the sea will have a lot of blue samples in it, but little red. A picture of a forest will be dominated by green. The sample process itself would already have to use interpolation which is not desirable due to the introduced error.

sample strip

A better alternative is to generate a hidden reference strip of samples. It consists of n^3 coloured pixels evenly spread over the range of possible colour values. n is the number of sample values per channel of the input colour space, as entered by the user when the profile is created. 3 is the

number of input channels. Look profiles are working in a PCS which both in case of XYZ and in case of Lab has 3 channels. The samples are generated by three simple nested loops iterating over the range of the three colour channels. The sample strip is generated when the colour correction dialog is opened and disposed of when it is closed.



True genius resides in the capacity for evaluation of uncertain, hazardous, and conflicting information.

Winston Churchill, English Statesman

The sample strip is affected by all the same manipulations as the image itself and thus provides the data to base the look profile on. Artificially generating samples instead of relying on the data contained in the image achieves a set of samples that is perfectly adjusted to the structure of the LUT. The reference strip contains precisely the colours that will later be stored in the LUT. No interpolation is required during the sampling process.

6 Evaluation

This chapter describes the steps taken to evaluate the usefulness and correctness of the implemented system.

As before, the chapter is divided into two parts, one for colour management and one for colour correction. Both parts start by giving a justification of the interface design. A full user-based evaluation was unfortunately not possible due to reasons hinted earlier. Several attempts were made to receive feedback on the CinePaint mailing list, in particular on the part of colour management which is hardest to test due to the difficulty of creating reliable profiles. These attempts had little success, however.
The chapter proceeds by outlining the tests that were done to assure the correctness of the implementation. In the case of colour correction this includes a comparison of the alternatives in the implementation, namely Gaussian curves and Bernstein polynomials and HLS and L*a*b*.

6.1 Colour Management

6.1.1 Interface Justification

To develop an understanding of what the user interface had to provide, existing systems with colour management functionality - Apple ColorSync, Adobe Photoshop and Scribus - and a study on colour management workflows served as foundations ([13]).

The definition of an image, workspace and display profile has become a de facto standard that is used in all colour management systems. The options for default behaviour are mainly inspired by Adobe Photoshop.

The fairly large number of settings to define the default behaviour. may appear redundant at first. They do, however, allow the maximum flexibility. Two examples may illustrate this:

Example 1: The user has a whole range of images that she knows to be in colourspace X. Her working colour space, however, is Y and she would like the images to be in Y before she edits them: She selects X as her default image profile, selects Y as her workspace, sets "when opening image" to "assign default profile" and makes CinePaint convert automatically to her working space. Whenever she now opens one of her images, it will automatically be converted into her working colour space and you can edit it.

Example 2: The user has a range of images in different colourspaces. He needs to edit them all in his workspace Y. In this case, he selects Y as your workspace, sets "when opening image" to "prompt for profile" and again, makes CinePaint convert automatically. Now, whenever he opens one of the images, he will be asked for the appropriate profile and CinePaint will convert the image to his workspace automatically. If he only needed some of the images converted, he could make CinePaint prompt you each time on whether to convert.

Since colour management can be a serious performance drawback it made sense not to force it onto the user, but to leave it at her discretion to switch it on and off.

6.1.2 Testing the Implementation

the cache

The most critical part of the implementation is the caching mechanism which was tested by adding debug messages specifying whether a profile or transformation was loaded from the cache or from from disc. The swapping of transformations into temporary files could easily be observed from the creation of files in the /tmp directory while the program was running. From the /tmp directory and the debug messages the correct operation of the cache was tested.

Opening a great deal of different pictures with the same profile combination never resulted in more than one transformation being created. Reducing the maximum number of unused transformations in memory to one and continuously switching between images with different profiles (which causes transformations to be requested and released) resulted in transformations being correctly swapped between disc and memory.

cms in use

A test of the actual correction of the image display itself was difficult to set up, since effective colour management depends as much on good profiles as it does on the underlying system. Profiles depend not only on the monitor model, but also the graphics board, the ambient light and much more. Creation of good profiles is an art in itself and often requires special measurement hardware. One such device is available at the King's Buildings Graphics Centre (an *EyeOne* colorimeter) and I spent a day trying to work with it. I got nice results for some images, but did not manage to generate one generally useful profile.

Since the actual conversion is driven by littleCMS, however, it seems valid to argue that it is

correct, since littleCMS has been tested and tried in many projects and by many eyes over the last couple of years. Also, my implementation is already in use by one other CinePaint developer who focuses on colour-related developments and no quality problems have come up so far.

6.2 Colour Correction

6.2.1 Interface Justification

Colour wheels are a common colour correction tool. Their basic structure differs little from implementation to implementation. There are almost always three wheels. There is almost always an alternative between visual input using the wheels and numerical input.

The day spent colour grading at the BBC brought a few interesting new insights: in colour correction, there are often no easy recipes to follow. Which tool works best differs from situation to situation. This led to the conclusion that it should be desirable to have options to fine tune a tool rather than a single predefined setting which the developer considered best. Especially in cases where it is impossible to define a best option. This led to the design of the option dialog. It predefines the options that seem most suitable in most cases, but leaves the user the freedom to fine tune, as special cases arise.

Interface features that the BBC editor used a lot were:
1. Comparing the image before and after the manipulation
2. Entering values numerically. The colour wheel is a very intuitive tool, but sometimes precisely the same settings as before are needed to see whether they will work again.
3. Quickly applying an identical manipulation to other images without having to click through a forest of dialogs

Point 1 is satisfied in the interface design by being able to toggle the overall effect as well as the effect of each individual wheel.

Point 2 is provided for by the numerical input fields.

Point 3 can obviously be satisfied by look profiles. It somewhat overthrew the initial design decisions. Look profiles were initially intended to be stored in external files for a maximum of flexibility. Look profile files could be stored in any directory and applying a profile would correspond to selecting the file through a file selection dialog. Such a system, however, would not have been suitable for the quick and repetitive tests and checks that the editor was carrying out as he was applying previously used effects.

Still, it seemed important to make the file nature obvious to the user and to give him the freedom to store profiles in any directory and to move them between different workstations. The final design tries to combine the flexibility of files with a seamless integration with the interface. This was achieved by deciding that all profiles used at a given point have to be stored in a single directory. Profiles can then be presented in a simple list. This list can be kept on the screen at all time and allows to easily switch looks on and off. Predefining a directory as in the case of colour management profiles seemed more restricted than necessary, so the list dialog allows changing the current directory which makes the file nature apparent to the user.

When the user saves a new profile, the directory that is currently used in the look profile dialog will be offered as the default storage directory and the look profile dialog will be updated. If we then use the flipbook to move to the next frame we can keep the look profile switched on and see the same effect on other frames.

The fact that each display owns its own look profile dialog provides a maximum of flexibility. It allows using different displays for seeing an same image with and without a profile at the same time.

6.2.2 Testing the Implementation

A suitable test image for the colour corrector is a simple grey gradient from black to white. It consists of all possible lightnesses and therefore clearly shows the different ranges of shadows, midtones and highlights. Besides, grey has a saturation of zero leaving a lot of room for colour shifts.

weight functions

First, the weight functions were tested. Figure 5 of the appendix shows the results. It depicts a correction of 50% saturation towards red. Lightness is preserved in all cases, the used model is L*a*b*. You can clearly distinguish between the application to shadows, midtones and highlights. The figure also compares the two different weighting functions: Gaussian and Bernstein polynomials. You can see how the ranges are spread out more widely for Bernstein polynomials, but the results are smoother. Especially when shadows, midtones and highlights are applied together, the Gaussian functions add up too high in some parts.

colour models

Next, the correctness of the implementations of the colour models HLS and L*a*b* was considered. The correctness of the implementation of the HLS model can be seen from the correct display of the colour wheels in the interface. The correctness of L*a*b* is based on the correctness of littleCMS which it is provided by.

The models were further tested by observing their particularities as they are used to preserve lightness. Their behaviour coincided with the expectations which was further confirmation for their correctness: In many cases you will see that HLS causes the resulting colours to be too light, such as in figure 7 of the appendix. This is caused by the wrong assumptions about luminance that are made by the HLS model as was explained in chapter 2.5. There is a modified version of HLS called Hoffmann HLS [8] that is claimed to produce better results. However, implementing and testing it against ordinary HLS gave no noticeable improvements, so Hoffmann HLS was disregarded.
That L*a*b* models lightness more correctly can be best observed when correcting highlights. Increasing blue has a lot less effect than increasing green (see figure 8 of the appendix). This is because increasing blue would make the image darker a lot quicker, since intensities of blue are perceived a lot darker than equal intensities of red or green. In HLS the effect is always the same.

This is an example where the colour wheel is inappropriate as an intuitive interface element. Its even shape suggests that the magnitude of the effect was the same for any hue, as much for yellow as for blue.

The only oddity that was encountered was that white is affected when L*a*b* is used to preserve lightness. Using HLS, white is never affected, if lightness is preserved. White should be the lightest colour there is and cannot be changed without a loss of lightness. Any correction that preserves lightness should therefore not affect white.

I therefore manually computed some example conversions between RGB and L*a*b*. It turns out that after readjusting lightness in L*a*b* and converting back to RGB, some RGB values can end up with an intensity greater than the maximum. As an example, the attempt to keep lightness constant might result in a colour of (210,200,260) where 255 is the maximum intensity. In L*a*b* this will correspond to a perfectly well defined colour. In RGB, however, the 260 will be clipped to 255 which leaves the formerly white area with a blue cast. Defining ways of handling such exceptional cases would further improve the algorithm, but the perfect

colour correction algorithm would be a project of its own. Giving the user the option to choose between L*a*b* and HLS seemed a good compromise.

A conclusion is the place where you got tired thinking.

Martin H. Fischer, Scottish theologian

During the tests, performance turned out to be a serious drawback. On an older machine like the PII-300 that development was done on the user can virtually watch how the preview is rebuilt block by block after each change of settings. This is, however, a problem the colour corrector shares with other CinePaint tools and is mainly due to the underlying CinePaint rendering engine which is at the moment far less then satisfactory.

It was intended to run a final test against Adobe Photoshop as a reference implementation. It turned out difficult, however, to sensible compare the implementations, since the details of Photoshops implementation were obviously not unknown. Many factors influence the final outcome, such as the chosen ranges of the input parameters, the weighting curves, the colour models. Often there is more than one valid choice depending on the intentions. The test was therefore rather informal, but it was almost always possible to recreate a given manipulation in the other program, even if with slightly different settings.

Look profiles, finally, were simple to test. All that had to be done was to create two copies of an image. The first one was manipulated and the result stored. A look profile was created from the manipulation and applied to the second copy. The two manipulated images were inspected for equality using the colour picker tool to inspect individual colour values.

7 Conclusion

This chapter summarises the achievements of the project and presents directions for future work.

7.1 Achievements

The goal to equip CinePaint with a display colour management system has been fully met. CinePaint now is the second open source application after Scribus to provide such as system. The handling of profiles was realised as efficiently as possible assuring that no profile or transformation is loaded or precomputed more than once, irrespective of how often it is used. The only feature that is commonly found in colour management systems, but was not realised during this project is the handling of embedded profiles. It has been explained in chapter 4.2.1 that this is not a priority for work in film studios. However, the necessary infrastructure to load

embedded profiles into the CMS cache has been provided. Loading and storing profiles from various file formats such as TIFF and JPEG is currently being implemented by another CinePaint developer.

The goal to provide a colour wheels colour corrector has equally been met. It may even be argued that the corrector implemented is slightly beyond the original requirements, as it presents the user with a range of fine tuning options. The only non-satisfactory aspect is performance, but it has been explained that this is due to the underlying CinePaint rendering engine.

Look profiles have been implemented as anticipated. They are the truly novel part of the development in this project. It is a feature that is not commonly implemented yet in standard applications and is only starting to be explored by developers. Hopefully, look profiles will be a good foundation stone to develop new exciting CinePaint features on.

As an additional CinePaint feature outside the scope of this project, binary fixed point has been implemented as a new image data format.

7.2 Scope for future work

CMS The direction for future work on the colour management system has already been set. The next step will be to handle embedded profiles. An extension to printer colour management should also follow which would turn the current system into a full colour management system.

colour corrector For the colour corrector, there is a wealth of possible extensions. The logically next step will most likely be the integration with the curves tool that is already hinted in its interface. Importance should then be given to tackling the performance issue. The most exciting colour manipulations become a lot less satisfactory, if they cannot be previewed interactively. This will, however, require thorough refactoring of the CinePaint rendering engine. To give just one example: Even though an image may be displayed on the screen at only a fraction of its real size, every preview involves processing the full-sized image data instead of a down-sampled and adjusted version.

Other possible extensions to the colour corrector are:

- a split screen which allows to show half the image before and half the image after the transformation to enhance the possibilities of visual comparison.
- an automatic matching feature. The user selects two colour samples from two different images. The algorithm will automatically adjust the colour balance of the two images such that the two selected samples match. This can be used to adjust different shot that are cut into one scene
- a visualisation of brightness ranges. To visualise which parts of the image are considered as shadows, midtones and highlights the pixels of the given range or highlighted or all other pixels are blacked out.
- a visualisation of the image's colour data. For every colour that is contained in the image, on point will be plotted in the appropriate position on the colour wheel. This gives a good indication of whether there is a strong imbalance towards one particular hue in the image that may have to be corrected.
- An appropriate extension or even replacement of colour wheels. It has been mentioned before that colour wheels are colorimetrically incorrect. Often this is irrelevant when they are only used as interface elements. In some cases, however, they will give wrong

indications to the user. An example was in chapter 6.3: Adjusting the highlights of an image towards blue will have a lot less effect than adjusting towards yellow, if lightness has to be preserved. The wheel, however, indicates that the adjustment was the same for all hues and thus conveys wrong ideas to its users that can lead to difficulties and unexpected behaviour.

look profiles

An extension to look profiles was already mentioned in chapter 3.1.4. There should be the alternative of storing the exact input values to the manipulation functions instead of only sample data obtained from the image before and after the manipulation. Their use would be slower, but allow the reapplication of profiles without any loss of quality caused by interpolation.

Other areas of improvement will likely involve the handling of profiles such as the possibility to enter meta information when the profile is created or to arrange profiles into categories in the look profile dialog.

It can only be hoped that CinePaint will carry on in its promising development and eventually become the professional and robust tool it is aspiring to be. It is a very ambitious project with a very friendly group of developers that to work in has been an interesting experience and a great pleasure.

References

Colour

[1] Color Models & Color Wheels, from: Watercolors, Bruce MacEvoy, 2002, http://handprint.com/HP/WCL/color6.html

[2] Colormaking Attributes, from: Watercolors, Bruce MacEvoy, 2002, http://handprint.com/HP/WCL/color2.html
The watercolors website contains an enormous wealth of information about colours from an artistic point of view. The two pages quoted contain a detailed overview of colour models and their origin and a good description of the colour making attributes.

[3] Computers & Colors, Bruce Fraser, chapter 2 in Real World Color Management, Peachpit Press, 2003, also at: http://www.creativepro.com/story/feature/19500.html
a highly readable introduction to colour encoding, its origins and development, and the problems of device dependency

[4] Frequently Asked Questions about Color, known as "The ColorFAQ", Charles A. Poynton, http://people.ee.ethz.ch/~buc/brechbuehler/mirror/color/ColorFAQ.pdf

[5] Frequently Asked Questions about Gamma, known as "The GammaFAQ", Charles A. Poynton, http://people.ee.ethz.ch/~buc/brechbuehler/mirror/color/GammaFAQ.pdf
The standard reference material on colours in computing.

[6] Computer Graphics: Principles and Practice, Foley, van Dam, Feiner and Hughes, 2nd ed, Addison-Wesley
The standard book on computer graphics. Chapter 13 is on colour and colour models. It contains the HLS-RGB conversion algorithm used for the implementation of the colour wheels colour corrector

[7] Colour Space Conversions, Adrian Ford and Alan Robert, 1998, www.poynton.com/PDFs/coloureq.pd
Descriptions of many colour spaces and colour models and conversions between them

[8] Color Order Systems, RGB, HLS, HSB, Gernot Hofffmann, www.fho-emden.de/~hoffmann/hlscone03052001.pdf
Introduces Hoffmann HLS, a revised and presumably improved version of HLS. For the application in this project, no significant improvement could be observed, however.

[9] efg's Color Reference Library, Earl F. Glynn II, http://www.efg2.com/Lab/Library/Color/index.html
Probably the most complete link and reference collection on colours, colour theory and colours in computing that there is.

Colour Management

[10] Why Colormanagement?, James C. King, Adobe Systems Incorporated, http://www.color.org/whycolormanagement.pdf

[11] Inside Colormanagement. The latest on what color management is, how it works, and what tools are available, David Broudy, Electronic Publishing, 2001, http://www.rit.edu/~lwlppr/trends/ColMan.pdf
Two introductory texts explaining the necessity and functionality of colour management systems to users. I prefer King's text. It has more depth. Broudy's text is slightly polemic.

[11] Color Management and Color Science: Introduction, Norman Koren, http://www.normankoren.com/color_management.html
Another introduction to colour management, but with more focus on explanations and aspects of colour sciene. A good link collection.

[12] Color Management, Jonathan Sachs, 1999-2000, Digital Light & Colour, http://www.dl-c.com/Color Management.pdf (there is a space between "Color" and "Management")
In-depth explanation of a full colour management setup, with focus on MS Windows and Digital Light & Colour's Picture Window Pro

[13] Colour Management Workflow Analysis, Søren Winsløw, 2000, The Graphic Arts Institute of Denmark, http://www.color.org/sorenwinslow.pdf
An analysis of how nine different newspapers incorporate colourmanagement into their daily work

The ICC standard

[14] File Format for Color Profiles (Version 4.0.0), Specification ICC.1:2001-12, 2002, International Color Consortium, http://www.color.org/newiccspec.pdf
the official specification of the ICC profile standard

[15] Building ICC Profiles - the Mechanics and Engineering, Dawn Waller, 04/2000, International Color Consortium, http://www.color.org/iccprofiles.html
An introduction to reading and writing ICC profiles. Has code samples and is generally more readable and informative than the specification itself. Contains a good introduction to the different processing models ICC profiles offer.

[16] Colour Calibration for Colour Reproduction, P.Emmel, R.D.Hersch, Ecole Polytechnique Fédérale de Lausanne, Switzerland at IEEE International Symposium on Circuits and Systems, 05/2001, Geneva Switzerland, http://diwww.epfl.ch/w3lsp/publications/colour/ccfcr.pdf
contains a good summarized description of ICC profiles in the first half of the article

Colour Correction

[17] Color Correction in Final Cut Pro, Steve Martin, Ripple Training, twosense.net/macworld_sf/pdfs/C65.pdf
slide show of the colour correction tools available in final cut pro. Since it is slides, it has little depth, but gives a reasonable first overview of the scope of colour correction in film

[18] Color Correction/Finishing in FCP, Andrew Balis, Kenneth Stone Photography, 09/09/2002, www.kenstone.net/fcp_homepage/cc_legal_fcp3.html
step-by-step explanations of colour correction in Apple Final Cut Pro 3, a film editing package. Even though product-specific, it is a very good overview of colour correction explaining many general concepts

[19] Color Correction By the Number
An introduction to the principles of colour correction using Adobe Photoshop. Claims that even colour blind people could colour correct which may be a little far fetched, but it still gives a very thorough and understandable basis.

[20] Photoshop - Restoration & Retouching, chapter 4: Working with Colour, Katrin Eismann, New Rider, http://www.digitalretouch.org/2edition/downloads/extras/PSRR_2ed_ch4_colorcorrect.pdf
A broad and thorough overview of all the tools Photoshop offers for colour correction and which to use for what

[21] Avid Color Correction, User's Guide, 2003, Avid Technology Inc., available from http://support01.avid.com/support/ProdRef.nsf/DocsR3?ReadForm&Start=1&Count=30&Expand=4.6
How to perform colour correction using the tools provided in Avid Symphony, the industry standard video editing suite